Universitat Politècnica de Catalunya
Departament de Lenguatges i Sistemes Informàtics

Master Thesis

styleBook - an interactive style social network

author: Patryk Kaczmarek
ponent: Pere Botella López
director: Gabriel Anglada

Barcelona, 2009

Abstract

In this thesis I present *styleBook*, an interactive reserve system for hairdresser salons extended with a social network functionality. The main aim of the application is to facilitate management of reserves and provide a tool that would improve your look. Once a user finds a haircut that is suitable, then it lasts seconds to find appropriate hairdresser salon located in the neighbourhood and to book a hairdresser. After a visit hairdresser is able to upload a picture of a haircut that was just done.

The main stimulus to create *styleBook* was lack of system that would enable reserves to hairdresser salons through the Internet in a global way. Furthermore hairdresser salons have no effective way to manage information about their clients. Nowadays there is also no social network related strictly to style matters. A target group for *styleBook* are females in age 15–30 years old from Spain and Poland.

Thanks

I thank Natalia Szwedowska, Gabriel Anglada and MASA-Ingenieria company for their help and contribution into this project. With their cooperation it was possible to improve the graphical design of *styleBook*, consult doubts that occurred during development of the project and receive any support that was currently desired.

Contents

Chapter 1

Introduction

The first chapter contains short abstract introduction to the subject of computer-aided hairstyle management. Moreover this part describes main aims of the project and their justification according to the project scope. Finally logical structure of the paper is presented together with a description of team task delegation.

1.1 Foreword

The rapid development of information technology in the recent years influences a big variety of fields. Without any doubts computers become indispensable in many disciplines, from industrial purposes to domestic. The advantages that we get outclass dwarf, older and more traditional methods. The technological advances provided, have a major impact on development of human civilisation.
The biggest advantage lies in a simple synergy – the combination of a human mind and processing abilities of machines. This creates unlimited possibilities and boosts a progress and as a result simplifies our lives.

The following master thesis[1] is a project that consists of creation of a reach Internet application. According to master thesis normative of Master in Computing, this work is a "Tesi modalitat b" also called a technological project.

1.2 Motivation

The idea to create this project came up in response to the requirements of the Spanish market. After short analysis of existing applications and numerous consultations with hairdresser salons, we decided to create a web service that would enable easy reserve system. To make it possible I have improve a communication layer between clients (users surfing in

[1] Master thesis - later on called in a simplified way: *thesis*

the Internet) with hairdressers. The application has been divided into 3 modules for each kind of user: a client, a hairdresser and a boss.

1.3 Thesis structure

This paper is divided into 7 parts, the first one being an introduction that describes briefly the main idea of *styleBook*. The second chapter - problem analysis contains useful definitions and a study of hair salon visits. Further more this part does a short analysis of a market with focus on existing applications and explains social network phenomena. The solution model development process is depicted in the third chapter. This part describes the actual solutions of certain challenges encountered in this work. It also describes the assumptions made during the design stage as well as the specific ideas used. The fourth chapter consists of implementation concepts and technical documentation. The use of particular solutions is described and justified both in macro- (technology, libraries etc.) and micro- (design patterns, specialized classes etc.) scale. A detailed application description is located in the fifth chapter whereas the sixth chapter incorporates project testing methodology. The seventh chapter contains conclusions drawn during project realization as well as a short summary of accomplishments. Not only does this part denote research made in fields not related to computer science, but it also includes the comments and concepts for the future improvements or feature supplements. This part comprises also possible project improvement suggestions.

1.4 Team task delegation

Whole project has been divided into series of small tasks that have been delegated to an appropriate person. Each task was elaborated in a specific phase. This approach was aimed to reduce bottlenecks and avoid interpersonal dependencies, i.e. situations when person A would be waiting for person B that needs to finish some job. Below general task delegation is presented:

Patryk Kaczmarek

- problem analysis
- architecture selection
- implementation
- testing
- deployment
- master thesis creation

Gabriel Anglada

- problem analysis
- creation of *Top Menu* written in *Flash*
- testing

Natalia Szwedowska

- problem analysis
- graphical design
- testing

Chapter 2

Problem analysis

This chapter begins with a definition of hairstyle, hair salon and social network. Next existing applications and various reserve approaches are studied pointing out possible improvements. At the end short history and examples of famous social networks are provided together with explanation of this phenomena.

2.1 Definitions

2.1.1 hairstyle

> *A hairstyle, hairdo, or haircut refers to a styling of head hair. The fashioning of hair can be considered an aspect of personal grooming, fashion, and cosmetics, although practical considerations also influence some hairstyles. Hairstyles are also influenced by various subcultures [2]*

Simply, it is a way in which a hair is cut and arranged. There are many different haircuts and categorising some of them might be a difficult task. Generally hairstyle characteristics can be divided into the following groups: gender (male, female), hair type (thin, thick, dry, etc.) cut type (straight, curly, wavy), hair texture (short classical, short modern, medium, etc.), hair colour (blond, brown, black, etc.), fringe type (straight, sideswept, panelling), highlights(done with cup, foil, brush).

2.1.2 hair salon

Is a company that provides services related to hair-cutting, styling or colouring. The more general term is a beauty salon which deals generally with cosmetic treatments including also skin health, facial aesthetic, foot care, aromatherapy, oxygen therapy, mud baths, and many other services.

There are 2 different types of hair salon. First one called walk-in salon requires no previous appointment, a client just walks in and gets a service. Second is a full-service, which requires a previous appointment. The quality, price, specialisation differ among hair salons affecting customers' choice.

2.1.3 social network

According to Wikipedia [2] a social network (*SNS*) is:

> *a social structure made of individuals (or organizations) called "nodes," which are tied (connected) by one or more specific types of interdependency, such as friendship, kinship, financial exchange, dislike, sexual relationships, or relationships of beliefs, knowledge or prestige.*
>
> *Social network analysis views social relationships in terms of network theory about nodes and ties. Nodes are the individual actors within the networks, and ties are the relationships between the actors. The resulting graph-based structures are often very complex. There can be many kinds of ties between the nodes. Research in a number of academic fields has shown that social networks operate on many levels, from families up to the level of nations, and play a critical role in determining the way problems are solved, organizations are run, and the degree to which individuals succeed in achieving their goals.*

Summing up, social network is a group of people that have something in common. This relationship is generally presented as a friendship relation or belonging to a group. User searches for friends or people that seem to be interesting and requests a confirmation of a friendship. After accepting the request by both sides, users start to share more information with each other. This results in easier communication and tightening of friendship.

2.2 Studying existing applications

This section is an analysis of similar – somehow competitive applications that currently exist in the Internet. It is divided into 3 parts. First one describes applications that would be used by users in a global way. Next chapters identify country specific web-services.

2.2.1 Global scope

After carrying out a market analysis and interviewing both, people related to the style matters and ones chosen randomly, following conclusion was drown: there is no application working on a global scale, which would assemble people related or interested to style matters. This poor situation might be indirectly proved by typing *style social network* resulting in

no important result matches. Further more my master topic placed on a website of UPC [1], where no positioning techniques were applied, is positioned on 9th place.

2.2.2 Spanish or Catalan websites

This market is characterised by a tendency to use applications working on a global scale. There are very few applications that are written strictly for Catalan or Spanish people. That is why there is also no web service that would incorporate hair and style matters. In contrary there are at least three services associated with clothing questions and new ones are still emerging, e.g. *Bestuario*[2].

2.2.3 Websites for polish users

Polish web is full of proper applications released for needs of local users. The approach seams to be more "patriotic", i.e. almost each global portal has one or more national equivalents that are used commonly. For that reason I was able to find 1 application that is similar to *styleBook* to some extent. This service is called Styl.fm[3]. The general idea of social network is quite similar however it is more basic. There is also no reserve system functionality, which in reality in Poland would be difficult to introduce due to lack of appropriate equipment in hairdresser salons.

2.3 Studying hair salon visits

This part compares 2 case studies. First one explaining what actions people need to undertake in a classical approach. Second shows how *styleBook* improves and facilitates the chain of events required to cut a hair.

2.3.1 Reserve overview

Reserve is an act of setting up an appointment with a hairdresser. It requires establishing a contact with hair salon, selecting a hairdresser and choosing appropriate time. After booking is done, it can be modified or cancelled by the solicitor.

2.3.2 Classical approach - before introduction of *styleBook*

There are two basic approaches. First one could be called walk-in or colloquially "spontaneous". It takes place when a person suddenly - spontaneously decides when and where to go. No previous planning is done, i.e. calling or visiting a hairdresser salon. The main

[1] https://postgrau.upc.edu/

[2] *Bestuario* is a project, residing still in the development phase, which can be identified with clothing matters

[3] http://ikf.com.pl/

negative effect is the fact that there might be quite long queue that significantly increases waiting time.
Second approach, sometimes called full-service requires use of a phone or visiting the specific centre in person. Calling is also time-consuming and there might be problems with interpersonal communications, i.e. problems with foreign language, strange accents, noises on the line, etc. Furthermore there is no easy way to present current timetable to a client.

2.3.3 Conclusion and possible improvements

Both above approaches, even though working, are old-fashioned, consuming too much time and require increased effort from both sides - client and hairdresser. It is important to notice that nowadays majority of medium and big size hairdresser salons are equipped with computer. These machines are used generally to browse Internet, send emails and in some cases for accounting purposes. There is no system that would store information about clients, passed appointments and help in reserve handling.

2.4 Studying a social network phenomena

2.4.1 History

The idea of social network is very old, pointing back to times of ancient Greeks. Although the first recognisable social network called SixDegrees.com came up in 1997. Some portals, eg. dating sites containing partial functionality of SNS existed before. Some people claim that, eg. Classmates.com founded in 1995 was the first recognisable social network. In my opinion it was lacking the most important parts: creation of profiles or list of friends. A complete list of social networks launch dates is depicted in picture 2.1.

2.4.2 Famous applications

Facebook was marked in 2009 by the most used social network worldwide, evaluating amount of monthly active users. Starting from the academical scope (at the beginning available only to students of Harvard University) finally become available to anyone aged 13 and over. Nowadays is having more than 250 millions of active users. It is an enormous social network that provides an *API* (Application programming interface) to create customised applications. This interface and ability to upload unrestricted amount of photos is one of the reasons for such a big audience. Apart from it *Facebook* has the following features: *The Wall*, *Pokes*, *News Feed*, *Instant Messaging*, *Gifts*, *Marketplace*, which greatly help to gain the audience.
Apart from applications oriented for the global scale, there are many services specific for a given country. A good example can be polish social portal called *nasza-klasa* (Eng.“our-class”). This service was established in year 2006 by the computer science student. Nowadays

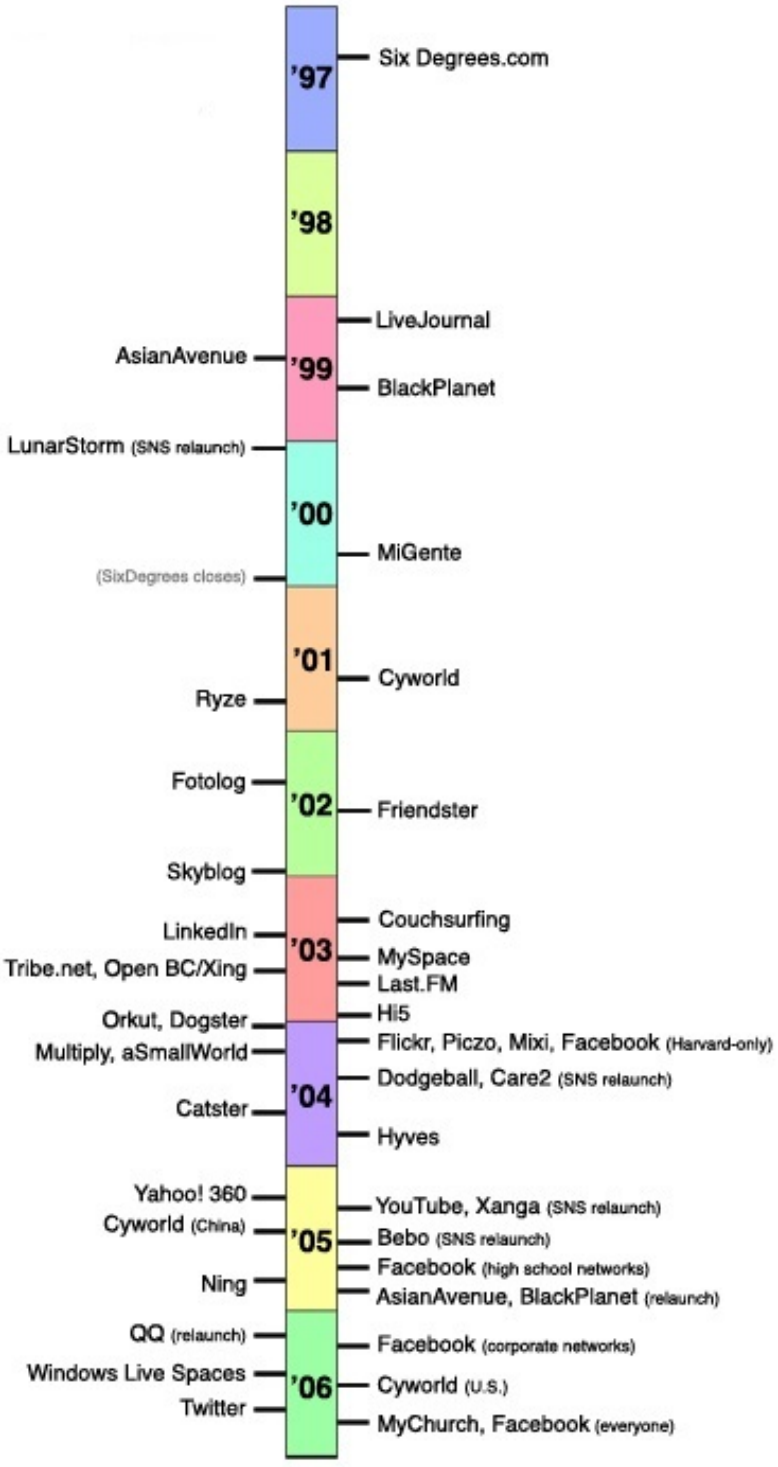

Figure 2.1: Launch dates of major social network sites
[10]

this service counts for 12 millions of real accounts, where great majority was created by poles. It is important to notice that Poland has only 38 millions of inhabitants, so relative percentage is extremely high. The idea is simple but smart, i.e. after registration you can find your schools and friends with whom contact was lost. This simplicity is the only reason for success of this service. There had already been more advanced and better applications like *grono.net* (2 millions of active users), but they did not become so popular. That is why selecting appropriate target audience is very important.

2.4.3 Six degrees of separation

Also known as "Small world problem" or "Human Web" is a notation strictly connected to Internet and social networks. It is an experiment investigating interconnections between different people, claiming that each person is at most 6 steps (friend of a friend relation) away from any other. This phenomenon has an influence on almost each area of knowledge. The closer an individual is to another individual, the more influence they have on one another. Social networks help to become aware of these relations.

2.4.4 YASNS

YASNS - Yet Another Social Networking Service is a common syndrome nowadays. Already there are plenty of social networks linking people from almost all social environments sharing many different interests. This is a reason for failure of many new projects that do not introduce any original functionality,lack appropriate advertisement strategy or target inappropriate audience.

Chapter 3

Solution Model

3.1 Studying the process

Understanding the reserve system and behaviour of people using social networks was crucial to develop an appropriate model. False assumptions would lead to partial or complete system failure. That is why we have visited several hairdresser salons and went to *Cosmobelleza Fair* that took place in Barcelona to understand better the whole process.

3.1.1 General model

The schema below presents all steps that are required to make a classical reserve.

Figure 3.1: General reserve schema

First a person has to decide what kind of a hairstyle would be the most appropriate. It might be either the same as previous one, which simplifies hairdressers task, or a completely

different one. Later on person has to either book a visit or go directly to the hairdresser. Once seated on a chair, the client has to describe a desired look. There are many tips and tricks that facilitate choosing a good hairstyle making hairdressers task easier (see apendix A for more details).

3.1.2 New approach with *styleBook*

Our solution aims to revolutionise way of making reserves. The main idea is to provide the application that would enable making an appointment with help of an interactive calendar. After selecting required date and time user can add a picture and description of a desired hairstyle. In this way a hairdresser is aware of all requirements and misunderstandings related to final result are eliminated to a great extent. General model and model improved by *styleBook* can be found in fig 3.2.

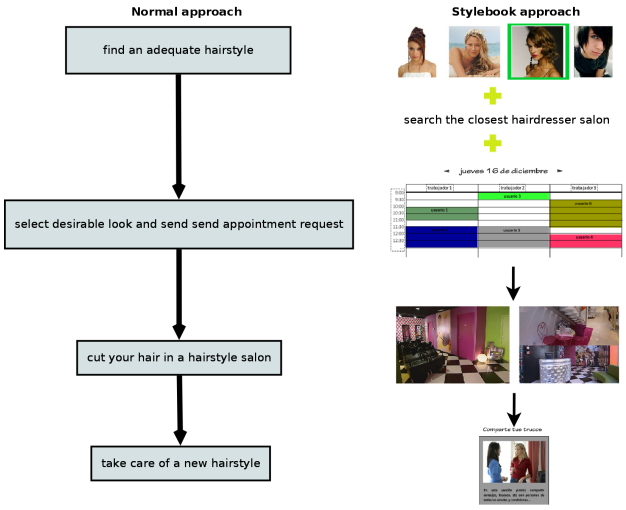

Figure 3.2: Reserve with styleBook

Making a reserve is divided into the following phases:

- hairstyle selection - is a moment when a client makes decision about preferred hairstyle. User can browse our database of photos and select one that seems to look the best. To make this task easier *styleBook* is equipped with Ranking and Search by Tags applications. The first one lists the top rated hairstyles and enables to filter-out unwanted results. Search by Tags is a place where user can view all hairstyles according to selected tags. Further more user can browse new looks or copy a hairstyle from one of friends.

- making a reserve - user select dates and times that are the most suitable in order of preference. Then waits for confirmation or other time suggestion returned by a hairdresser. There is no need to select multiple dates, but then a risk of rejection

increases significantly. We have chosen such solution, because a client might not be aware of exact time required for the visit. Then if user selects only one date and underestimates time, rejection might appear. In such case this user would have to wait for hairdresser reaction and repeat a reserve process. Such way of communication might be a bottleneck of application. Despite the fact, we assume that clients are more or less aware of required time.

- getting a photo from a hairdresser - user might receive (depending on a hairdresser) photos of the new hair style. These pictures should be taken after a finished cutting from different positions. Preferably from back, front and side to fully visualise the new look.

- social network part - other users can comment your new look, give some tips and advises how to take care of a new haircut. A user can also add selected photos to our ranking system. This part contains functionality of a standard social network. Permits email exchange, uploading photos or rating pictures of others.

3.2 Project management

A project management life cycle is composed of five phases: planning, design, implementation, verification and maintenance of a software product. It joins the life cycle of the system from definition of its requirements to the termination of its use [1]. Most important phases in our project are described below in the following section together with estimated risk management. Identifying the objectives and ensuring that they are met is very important to create a successful project.
In general there are some key-characteristics that describe a project.

- good planning is required
- specific objectives need to be met
- work is carried out in several phases
- work is carried out for external client
- the project has an absolute lifetime
- non-routine tasks are included

Having in mind this characteristics, I have focused on avoiding some common pitfalls that occur during software development carried out by students [14]. Below ideas are presented about different steps of project development.

3.2.1 Software engineering

Selection of appropriate programming design technique was a crucial task. The main idea was to create a primary prototype (for requirements of master thesis) and then continue developing and improving it in a bigger group of programmers. That is why a mixture of two models have been used. For requirements of this master thesis, serving as the first prototype, waterfall model suited perfectly.[1] Having a well defined architecture and user interface established in a detailed way was very crucial. I have also considered using a more flexible, iterative approach, but there was enough certainty about how a system was to be implemented. Further more it was important to divide a project into several, logically arranged phases. Thanks to this approach each person could previously reserve required time for the project development.

In general we had 8 months of a time to develop the project. Time proportions were allocated according to figure 3.3.

Analysis The analysis phase started in November and lasted till beginning of December. This phase involves collecting information about a customer needs and defining the problem that has to be solved. We have arranged several meeting. As a result the draft of a project was created.

Design This phase was the most important in our case, because radical changes of project architecture were almost impossible to introduce with time. That is why it lasted two months, from December to beginning of February. Simultaneously I have been learning basics of required technologies.

Coding This phase was the most time consuming and lasted 6 months from February to August. During this time I have set up all configurations and done programming part. Further more I was studying the required technology in details.

Testing and installation 2 weeks of August have been spent on testing, evaluating and installing the project.

Documentation 3 weeks have been dedicated to sum up all documents, starting in August and ending in September. I tried to provide a detailed, but consistent information to current and future team members.

As said before to suit this project to our needs I have selected a waterfall approach. There was also a possibility to use incremental approach, but it was not the most appropriate for this project, because it is to chaotic.

Below there are listed the advantages with a short explanation and justification if required.

[1] In this thesis I will be only referring to this model and to the scope of master thesis.

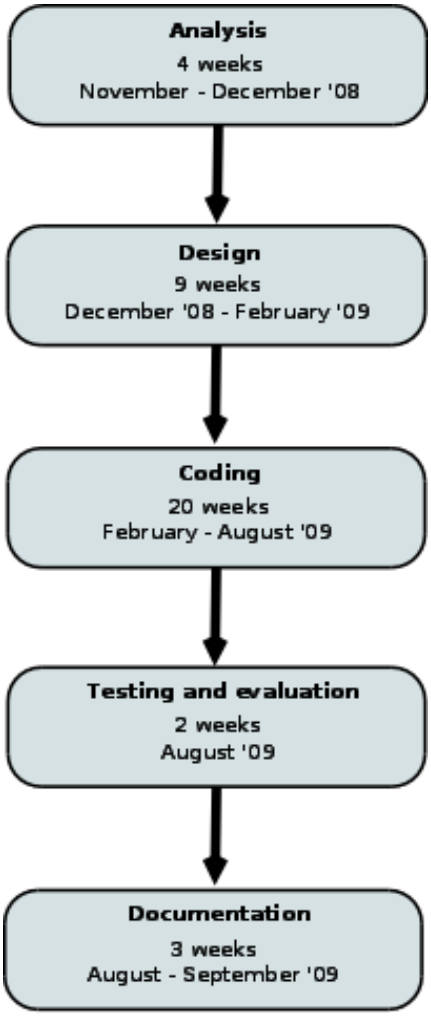

Figure 3.3: Work Schedule

- the staged development enforces discipline - the progress can be constantly measured. This helps in keeping high motivation in a team. It is important to notice that *styleBook* is not the only project developed at one time.
- minimal wastage of time and effort - due to emphasis on requirements and design.
- improved quality - because of getting design and requirements first
- possible errors corrected at early state of development
- aid efficient knowledge transfer when team members are dispersed in different locations. As we were working in a international team this advantage was very crucial.

There are also disadvantages of such approach

- customers do not know what they want, thus they would like to see some part of application first - not a disadvantage in our case, because it is a proper project and we were completely aware of the requirements.
- estimating time and cost is extremely difficult - this disadvantage did indeed affect the project. Although more time than required was assigned, some delays could be observed.
- designs cannot be translated into real products - in fact some redesigns were required but they did not influence negatively the initial idea.
- implies a clear division of labour - is said to be inefficient and unrealistic in majority of companies. In our case it was not an issue, but an opposite. There were only three persons with different specialisation, so each could concentrate on a different task.

3.2.2 Risk analysis

To avoid excessive risk during the project development, the following table has been prepared to identify possible risks:

Event	Probability	Importance	**Seriousness**
Technology compatibility	5	8	**40**
Software bugs	5,25	7	**36,75**
Modules integration	4,5	8	**36**
Implementation delays	7	6	**36**
Database design	4	8	**32**
Network programming issues	3,75	7	**26,25**
Web Service issues	3	8	**24**
Usability problems	2	8	**16**

Table 3.1: Risk overview

In above table each team member evaluated a probability rating (1–10) and a seriousness of a risk (1–10). The equation 3.1 shows how overall score is calculated using a simple formula:

$$Seriousness = \frac{1}{n} \sum i = 1^{n} Probability_i . \sum i = 1^{n} Importance_i \tag{3.1}$$

To obtain the final seriousness factor, first an arithmetic mean for each problem is found (according to each team member rating). Than this mean is multiplied by importance factor estimated by myself.
This approach allowed us to find out most serious risks and take preventive measures. Some of them are presented below with a short explanation.

Technology compatibility In order to prevent technology issues I have spent a lot of time in design phase taking care of selecting appropriate technologies. Furthermore consultations with experienced programmers were undertaken.

Software bugs To accomplish this task, only well known and tested components and technologies were used.

Modules integration Each module was written as a stand alone application requiring only a global *Model Locator*4.3.2. I was trying to avoid tight coupling between the main application and different modules.

Implementation delays To keep work in progress, systematic meetings were scheduled.

Database design Database was designed with a great care. Furthermore it was improved during the development of the project. Help of more experienced database programmers was very useful in field of database design, solving algorithmic problems and query optimization.

3.3 Implementation concepts

During the analysis stage various architecture concepts were considered. Finally I have chosen the *Flex* technology that imposed an architecture presented below.

3.3.1 Architecture

The architecture used for *styleBook* is a client-server. In our case it means that the server-side architecture of a Rich Internet Application typically exposes to its clients a Service API.

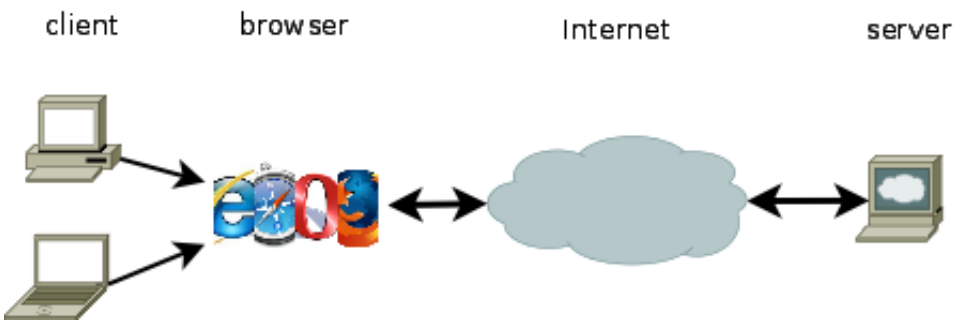

Figure 3.4: Application architecture

In this solution a user using browser downloads a client-side application that connects to the remote server. Depending on a user type, only required module is downloaded in order to limit traffic in the Internet (see fig. 3.5). User needs only a browser with installed Macromedia Flash Player (about 99% of machines have it already installed) . This software updates automatically, so that no further user interaction is required.

The application was divided into 3 stand alone modules that are communicating with main application through a *MainModuleLocator* - a singleton class used to exchange information between different components.

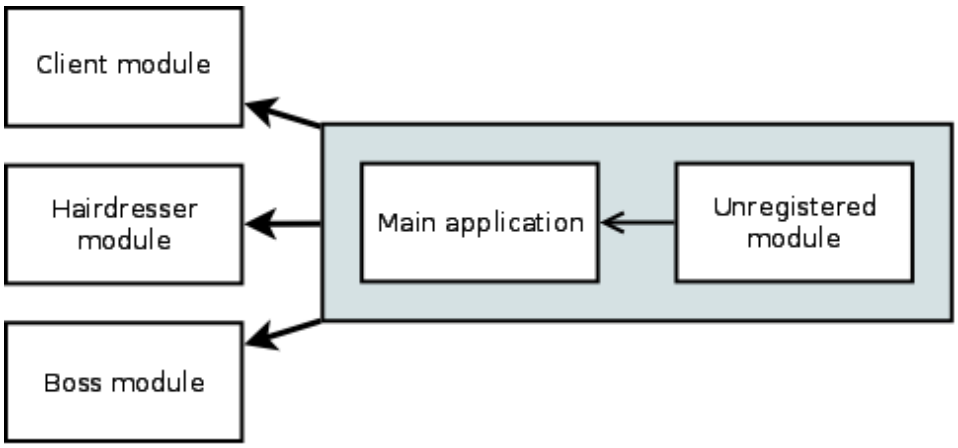

Figure 3.5: Main modules schema

The main application can use one of three following modules:

- *Client Module* - a module used by a regular or unregistered user. Serves as a social network extended with a reserve system. It is used to search for the closest centre, book a visit, talk and share information with friends.
- *Hairdresser Module* - used by a hairdresser to manage visits. After user has finished booking, a confirmation done by a an authorised personnel is required. This module is

also equipped in a component used to upload photos that are later available to clients.

- *Boss Module* - a place to control a salon. Using this module a boss is able to add new workers, modify company information and presentation layer or add news.

3.3.2 Testing and deployement

For requirements of this master thesis, *styleBook* project was tested on several local machines that use different operating systems. The code is being shared through the SVN server. In the future project would probably be uploaded to one or more servers with three specific environments:

- test environment - used to test current modifications. Available and visible only to programmers currently working on the project.
- QA environment - quality assurance serves as a testing environment available to the wider group of people. This server will be available to clients and testers that would share their remarks with programmers.
- production environment – code will be deployed only if all requirements are met in the previous environment.

Chapter 4

Implementation of the solution concept

This chapter contains a detailed description of implementation details. In the beginning used technologies, tools, applications and patterns are outlined together with a short justification of each choice. Afterwards a profound explanation of proposed solution together with functional and non functional requirements is presented.

4.1 Technologies

The main aim of this section is to provide a brief description of technologies used in the project. The focus is put on client-side *Flex* programming language, *Caingorm* and *AMF-PHP* frameworks and on server-side *PHP*, which play a crucial role in the project. Next *MySQL* database, *Apache* and *SVN* servers are described.

4.1.1 Flex

Flex is a software development kit released by Adobe Systems in year 2004. It supports development and deployment of cross-platform rich Internet Applications (*RIA's*that can be easily executed using *Flash Player*. The applications might be developed using Flex Builder 4.2 that has already all required compilers and debuggers integrated, or using an editor of your choice together with a stand alone Flex compiler. It emerged recently, because developers thought that Flash, is a programming language hard to learn and understand. They did not like using animation frames in creation of web pages. *Flex* code is composed of *MXML* - an *XML* based markup language and a dynamic *Action Script* language (at the end both are compiled to Action Script). *Flex* applications provide a stateful client that can be connected with majority of technologies serving as a server: *PHP*, *Adobe ColdFusion*, and *Microsoft ASP.NET* or *Java*. I have opted for *Flex* because it permits to create a *RIA*

with very little effort. In case of *styleBook* good graphical design and user interaction were very important matters.

4.1.2 PHP

PHP is a widely-used general-purpose scripting language founded by Rasmus Lerdorf in 1995. It is especially suited for Web development and can be embedded into *HTML* - serving a dynamic content. It runs on a web server that takes a *PHP* as an input and as an output returns a web page. This code can be executed on the majority of servers and there is no need to take care about the operating system. The popularity is the reason of small language complexity. It can be learnt rapidly without big time efforts. Although seems to be simple, there are plenty of applications created, thus its security is quite height. Also the awareness of most common programming errors and best programming practises is high among experienced developers. *PHP* is a perfect match for *Flex*. There are many frameworks that allow simple integration of both languages. Another important factor was the wide availability of web servers running with *PHP*.

4.1.3 Cairngorm framework

Cairngorm is the most popular framework for *Flex* developers available on the market. It enforces the use of well known patterns, enforcing the scalability and growth of applications[3]. In case of medium or big size application it is essential to follow some common guidelines and strategies of software development. *Cairngorm* permits different developers focused on the same task to work together at the same time. However some developers complain about its disadvantages. They say that too much code is repeated and they do not like an idea of global singleton class serving as a *Model Locator*4.3.2. I could have chosen one of different frameworks available to *Flex* that seam to enable faster programming, but in case of *styleBook* it was not a concern - it was not a project that had to be created in one week of time. It is important to remember that much more time is spent on testing and debugging than on creation of code. Further more *Cairngorm* has very good documentation and there are available many examples of code. See picture 4.1 for detailed information about *Cairngorm* architecture.

4.1.4 AMFPHP

AMFPHP is a free open-source *PHP* implementation of the Action Message Format(*AMF*). *AMF* allows for binary serialisation of *Action Script* (*AS2*, *AS3*) native types and objects to be sent to server side services without worry about the transport layer [20]. This framework allows an application written in *Flex* or *Flash* to communicate with a *PHP* server simply calling remote functions. The data is returned as an object, which is easily recognised in a client application. It could be easily integrated to work with a *Cairngorm*.

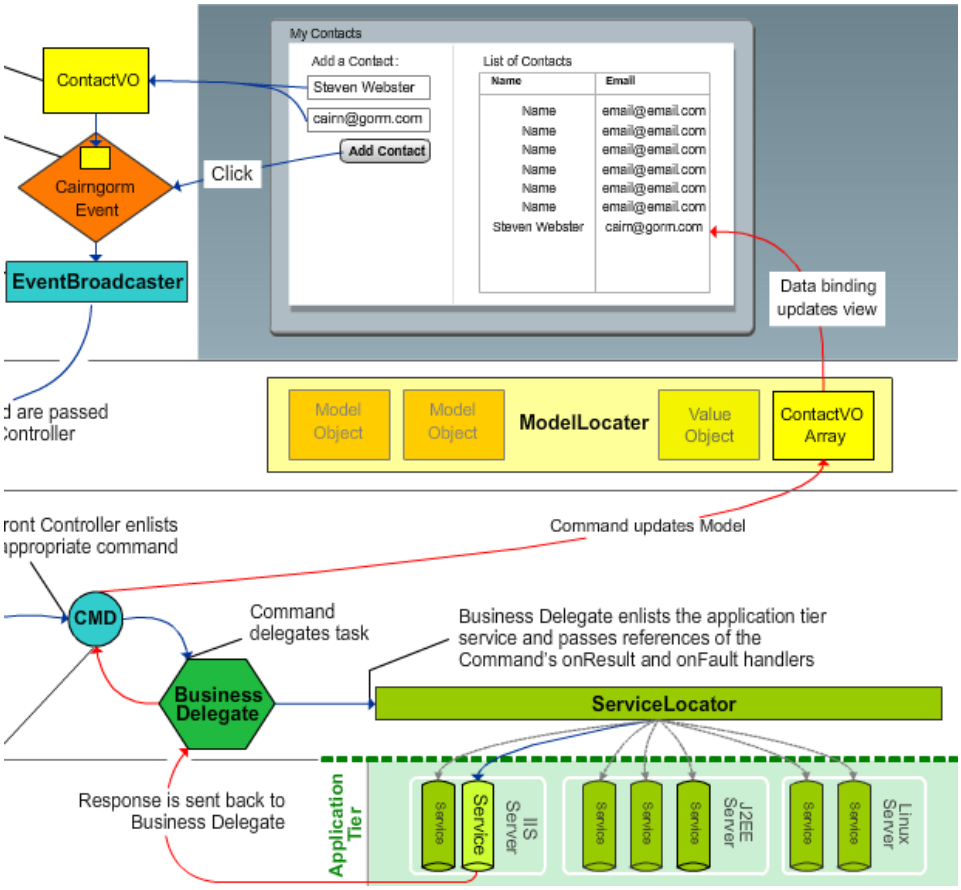

Figure 4.1: Cairngorms microarchitecture
[7]

4.1.5 MySQL

MySQL is the most popular database in the world, which is compatible with many different kinds of tables, e.g. *MyISAM* or *InnoDB*. It was created by MySQL AB in 1995 and recently purchased by Sun Microsystems[15]. This solution is open-source, nevertheless quite professional. The main advantages are high speed, simplicity and support from numerous programmers, which make it a perfect solution for our project. As an alternative we could have chosen many different kinds of databases, e.g. *Informix*, *Oracle*, *PostgreSQL*, or *BerkleyDB*. Unfortunately the majority of them are commercial.

4.1.6 Apache

Apache is the best known open-source web server that played important role in development of the *World Wide Web*. It can be run on plenty of platforms, from which the most famous are: *UNIX*, *Windows*, *Macintosh*, *Solaris* and *Novell*. *Apache* is composed of modules that can significantly increase its functionality. It supports *PHP*, *Perl*, *Python* and *Tcl*. We have opted for *Apache*, because it works great with *PHP*, is platform-independent and open-source.

4.1.7 SVN

SVN is a version control system. It is an open-source solution created in 2000 by CollabNet, used nowadays in many well known projects. It is the successor of *CVS* which was created for the development of projects by different programmers. The main goal of *SVN* is to keep current and historical versions of files, such as source codes, photos or documentation[8]. They are stored on a special remote repository (we are using a free of charge OpenSVN[1] server run by several enthusiastic students at National Taiwan University), which enables concurrent development of applications. Specific parts of *styleBook* project are shared between a designer and programmer thus it is important to keep the data actual. To achieve this goal, we have once more used an extension for *Eclipse* called Subclipse [2]. It is integrated with flex programming environment and provides basic *SVN* functionality: supports updates, commits, resolves existing conflicts and enables rollbacks.[6]

4.2 Tools and Applications

This section contains a list of only the most important tools and applications that were used repeatedly during project development. Other, less vital were used to prepare schemes, pictures, diagrams, i.e. *Dia*, *Gimp*, *Adobe Fireworks* , *Poseidon*.

[1] https://opensvn.csie.org/
[2] http://subclipse.tigris.org/

4.2.1 Flex Builder

Is an applications based on open-source *Eclipse* editor modified by Adobe to meet needs of *Flex* language. *Flex Builder* supports intelligent coding (*Action Script* editor), interactive debugging, visual design of user interface (editor for *MXML* and *CSS*). Furthermore it facilitates skinning and styling and can be integrated with Adobe Creative Suite 3. Apart from this special features it is also a normal *Eclipse* editor, which was invented by IBM and today is being developed by Eclipse Foundation. It is a platform independent editor with a very professional plug-in mechanism, which notably simplifies programming. Integration with *SVN* and database is really easy and intuitive just as using it. The other advantage is good code completion. Nowadays there is no reasonable alternative to *Flex Builder*.

4.2.2 LaTeX

LaTeX is a typesetting system designed to create technical and scientific documentation. It is used widely by mathematicians, scientists, engineers, philosophers, economists and students writing their thesis. I have used it in order not to worry about look and feel of this document, numbering, cross-referencing, tables, figures, page layout and bibliographies.

4.3 Patterns

Design patterns have been invented to simplify design, creation and maintenance of a code. The main advantages of pattern-based approach have been briefly presented in this section, mutually with specific implementation details [9]. The majority of this patterns were imposed by selection of *Cairngorm* framework. Others are used to improve simultaneous programming and make the code more extendible. To have better idea about patterns that are related to *Cairngorm* have look at figure 4.1 presenting its micro architecture.

4.3.1 MVC

The main assumption of the project is to provide appropriate level of customizability, systematization and adjustment. All this characteristics are provided by *MVC* pattern. It distinguishes three layers: model, view and a controller. The first is responsible for storing business data of the application. Model has no idea about existence of a controller or a view. The view component is only responsible for layer of presentation. In very rare cases it can directly call a model. In a normal approach, a controller is responsible for receiving and processing the data and returning results to the view layer. *Cairngorm* is a framework that completely incorporates a *MVC* pattern, but also does some modifications and many extensions to this model. This phenomena is described below in greater details. Each of this terms is strictly related to *Caringorm* and must be understood in order to develop applications using this framework.

4.3.2 Model Locator

Model Locator is a Singleton[3] that stores all the data of an application (preferably in form of Value Objects 4.3.3). This model is very useful in combination with *Flex* bindable objects. A change of a value of such object is at once reflected in all places where it was defined. The developer does not need to worry about updating many different elements of the view to reflect the changes of the model.

4.3.3 Value Object

Value Objects are very simple objects used to improve the readability of code. They normally are used to transmit data stored inside of them between different tiers of application. In *styleBook* I am mapping this object directly to a given value object residing on the server. That is why no conversion is required when objects are sent between a client and a server.

4.3.4 ServiceLocator

This class is a *Singleton* for service access. It might be extended through *MXML* in which all services are defined. A good manner and only reasonable is to define services on the server.

4.3.5 Front Contoller

Receives all *Events* and maps them to a special classes called *Commands*. Each *Command* provides one entry point, a method called "execute()". This method can be executed by any class, which might not be aware what it really does. If you want to add a new feature to a program, than you need to add new *Command* class here, which might also contain methods responsible for receiving the data from server.

4.3.6 Delegate

Delagate is responsible for business logic. It is a mediator between a *Command* that is calling it and appropriate *Service*. This layer seems to be the least necessary, but is left for better readability. In theory this functionality could be moved to a *Command*.

4.3.7 Observer

Observer belongs to a group of design patterns (not related to *Cairngorm* framework), for which the main purpose is to observe the change of an object in a program, in order to inform all methods that have previously been registered to it. However in majority of cases a simple data binding plays the same role.

[3] Singleton - a design pattern created to restrict instantiation of a class to one object

4.4 Data Model

As previously mentioned, to access a database I have used an open-source *AMFPHP* that uses an interface, which I created to read, write or modify data. More important that the way of accessing data was the idea to create a correct and complete model, which would guarantee the capability to represent all information relevant to the project. This model should reflect all the functional requirements. Apart from simple reserve data storage, system stores all data related to a social network, and a hair salon management. Below I have described briefly the most important characteristics of each class, more details might be seen on figures 4.2 (relations between different user types, messages and notifications) and 4.3 (general schema).

Currently the model is composed of 22 objects. Many of them have some properties in common, which will be described only once - below.

- *id* - an id used to identify an object
- *creation_date* - creation date of an object, stored in format [YYYY-MM-DD HH:MM:SS], e.g. 2009-08-10 22:13:46
- *modification_date* - date of modification, stored in the same format

This fields are updated automatically by the database, so no concern has to be payed.

4.4.1 User

As the name indicates this object represents a single user. It contains all the basic data such as name, second name, surname, second surname, email and password (stored as MD5[4]). Apart from it users city, phone and zip code are stored. After process of registration each user obtains unique confirmation code, which is required to activate an account. To make distinction between users types: a client, a hairdresser or a boss, type field is used. This field helps in distinction of specific users and aids in creation of child elements extending user object. There are also three variables that store information about current state of user. They help to check if user is blocked (the account can be blocked after improper behaviour), activated (if confirmation was done) or deleted (user information is stored on server to allow easy recovery later). At last info about last visit time and users default photo is saved in this object.

4.4.2 Friend

This object stores information about owner of the friendship (the person that sent a request), a requested friend and the state of a friendship. State might be the following

[4]In cryptography, MD5 (Message-Digest algorithm 5) is a widely used cryptographic hash function with a 128-bit hash value. As an Internet standard (RFC 1321), MD5 has been employed in a wide variety of security applications, and is also commonly used to check the integrity of files or to encrypt passwords[2].

- *pending* (*pe*) - the request was send by one side, but the second user did not still accept or reject friendship
- *accepted* (*ac*) - both users confirmed their friendship
- *rejected by owner* (*ro*) - the person that requested friendship changed the mind and rejected the invitation
- *rejected by friend* (*rf*) - the invitation was rejected by a requested friend
- *blocked by owner* (*bo*) - the sender of invitation wants to remove the friendship
- *blocked by friend* (*bf*) - the friend is fed up with a friendship
- *blocked by moderator* (*bm*) - the moderator blocked a friendship

I am also storing information about dates of sending and accepting the invitation.

4.4.3 Message

This object is composed of message, topic and dates describing send and receive time. To enable grouping of messages that are all responses to one message, an id of the first message is stored apart from ids of receiver and sender. There are also fields indicating the state of message. For sender I distinguish:

- *draft* (*dr*) - message saved for future editing
- *sent* (*se*) - message successfully sent
- *trash* (*tr*) - message moved to trash
- *deleted* (*de*) - message removed permanently

Fields for receiver are the following:

- *unavailable* (*un*) - message is still unavailable to view (might still be edited by sender)
- *available* (*av*) - available to see
- *viewed* (*vi*) - message already viewed by receiver
- *trash* (*tr*) - message moved to trash
- *deleted* (*de*) - message removed permanently

4.4.4 Notification

This class is used to transmit some information between two different types of users. A hairdresser might inform a client that the requested visit was accepted.

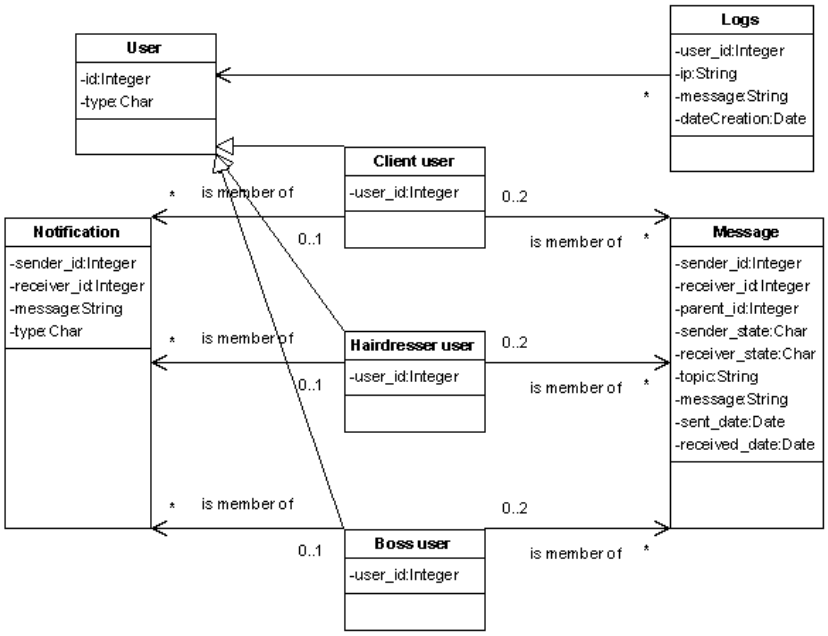

Figure 4.2: User communication model

4.4.5 Hairstyle Photo

This object represents a photo of a hairstyle together with information about its owner and about hairstyle it belongs to. It provides basic information like name, description, original name of the photo, path on the server and order. Further more it says if it is proper photo or uploaded by a hairdresser.

4.4.6 Hairstyle

Hairstyle might be compared to a gallery, which holds some amount of photos. It is described by the name and short description. It serves to group photos that belong to the same hairstyle. Each hairstyle has its specific access (public, friend, private), more detailed description concerning accesses can be found in section 5.3.2.

4.4.7 Tag

Tag is an object that characterises given hairstyle. Tags are divided into the following categories:

- sex / gender: female, male
- hair type : thin, thick, dry, oily, choppy

- cut type: straight, curly, wavy
- hair texture: short classical, short modern, medium, medium layering, long, long layering
- hair colour: blond, auburn, brown, black, red, fantasy mixture, white
- fringe type: straight, side swept, panelling
- highlights: cup, foil, brush

4.4.8 Vote

Votes are objects that characterise quality of given hairstyle. A vote has score from 1–10.

4.4.9 State

Used to inform other user about current state. There are three client states: available, gone, invisible.

4.4.10 Status

Displays a message that user has just entered. It is visible to other users that are browsing this profile.

4.4.11 Cities, Regions, Countries

This objects represent geographical location of given city, region or country. All are characterised by name, in case of region a shortcut is stored and country is associated with a language.

4.4.12 Company

Company is created by a boss. Its function is restricted, used only to manage details about itself and to add news. To create a hair salon, there is a need to add a Department object described below. Company is used to group up all and available departments.

4.4.13 Department

Department represents a hairdresser salon, it has a name, address and description. It is a place where hairdressers work. Each can be related to only one department in given time.

4.4.14 Hairdresser availability

This object represents time-table of given worker. It contains day of a week, start and end time. For one worker and one day there might be many such objects. We assume that someone may work in hours 8:00–12 and 14–18.

4.4.15 Preferred Reservation Time

Preferred Reservation Time is an object that stores time of planned by a client reservation. It contains the date (days after 1/1/1970), start time, end time (difference is measured in 30 minute periods, starting from hour 00:00), a hairdresser and preference. Each client can make three pre-reserves selecting free periods of time according his/hers preference.

4.4.16 Reservations

Reservation is one of Preferred Reservation Times. It is selected by hairdresser, which has in mind clients preferences and estimated duration of a treatment. It contains the same fields as previous object extended with a phone number and hairdressers comment. Furthermore it tells the time of confirmation and has a status of reservation (accepted or rejected by user).

4.4.17 News

News is an object used to store information about new activities in a company. It is created by a boss and contains text that should be displayed and a photo.

4.4.18 Log

Log object is used to register some of user actions. They are saved in database with the following fields: id of a user, ip of a computer some message and time the action took place.

4.5 Communication

This is one of the most complex and important subjects raised in the project. The choice of a communication architecture during the project design stage influenced selection of appropriate technologies. The application uses a client-server communication architecture, where *TCP/IP* present in the Internet is a medium used for data transmission. I had to decide on a way in which client connects to server and handles any exceptional situations and the results are presented below. Next a communication seen as an event flow is described. This is followed by explanation of asynchronous event processing and techniques used to allow a scalability of the application to various servers.

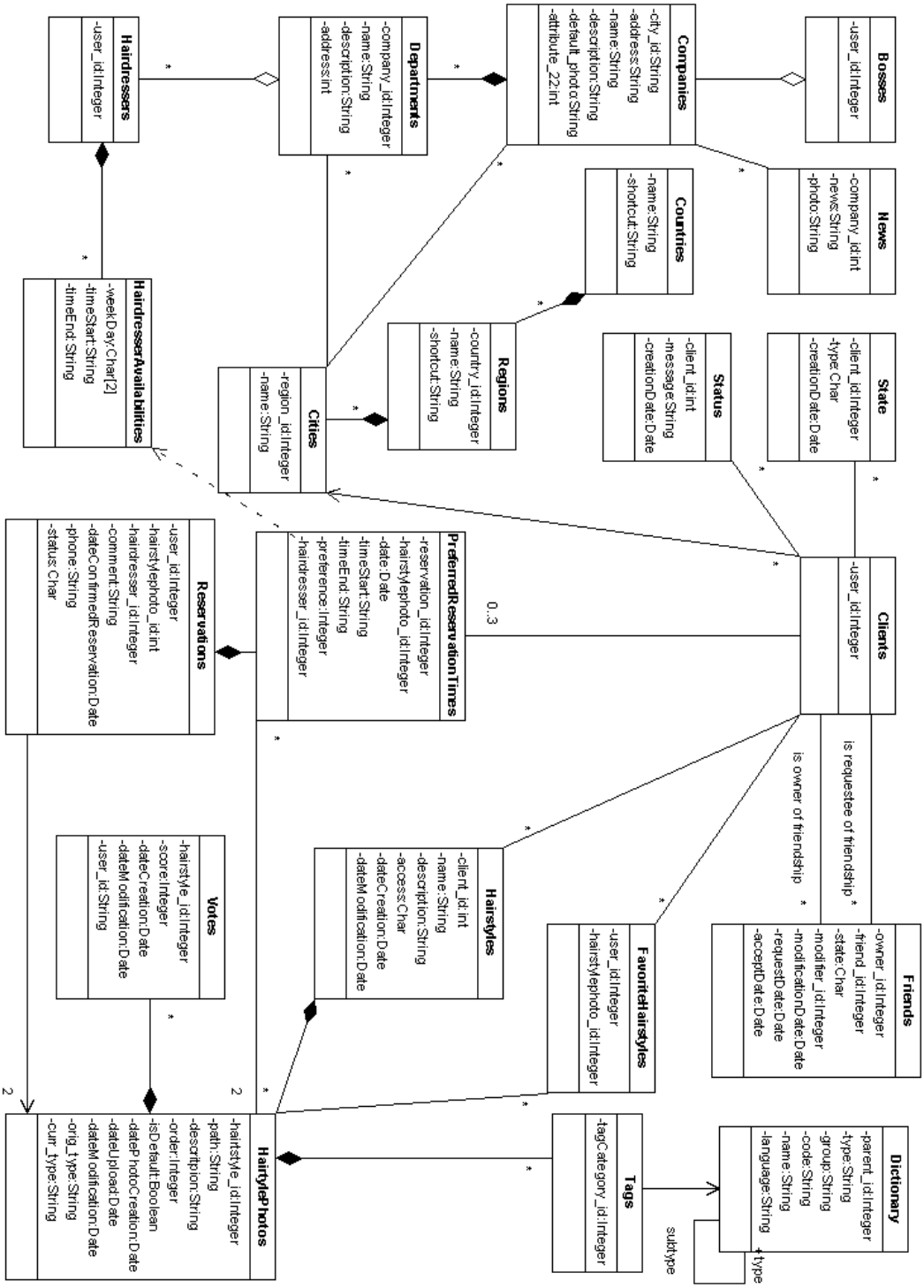

Figure 4.3: Main Model

4.5.1 Communication architecture

The biggest challenge was selecting an appropriate method of accessing server-side data, considering the fact that in can be solved in variety of ways. The easiest, probably the most resource consuming solution was to use HTTPService components. It is based on receiving a *XML* structure to the *Flex* application transported in HTTP response. Another solution was to use Web Service components. Flex can cooperate with applications that define their interfaces in Web Services Description Language 1.1. This solutions seamed to be too complicated and as a result still *XML* was returned.

Finally to communicate data between server and the client, I have opted for a more robust solution - Remote Procedure Call (*RPC*). The only thing that needs to be done in order to call a function on the server, apart from doing all configuration stuff, is to call a method on the local object, set a callback function and receive the result. A programmer does not need to worry about the implementation details - it is completely transparent. Trying to describe it shortly: the client serialises a request and sends it to a gateway. *AMFPHP* then automatically:

- Deserializes the request
- Finds the corresponding remote class
- Instantiates the class
- Performs security checks
- Calls the remote method using the specified arguments - does all the business logic that is required, calls a database if required.
- Serializes the returned data [20]

Next an object is returned to the client application, where it is used according to the needs. Most of the problems encountered during implementation of this part concerned integrating different technologies, creating appropriate configuration files and setting appropriate gateways.

4.5.2 Event flow

Is a way that classes interact with each other. The figure 4.5 depicts a typical information flow starting from the moment when user asks server for information and ending in the moment of its receival.

In details the flow is the following:

1. user does some action in the view, for instance clicks login button
2. when a button is clicked an event is dispatched

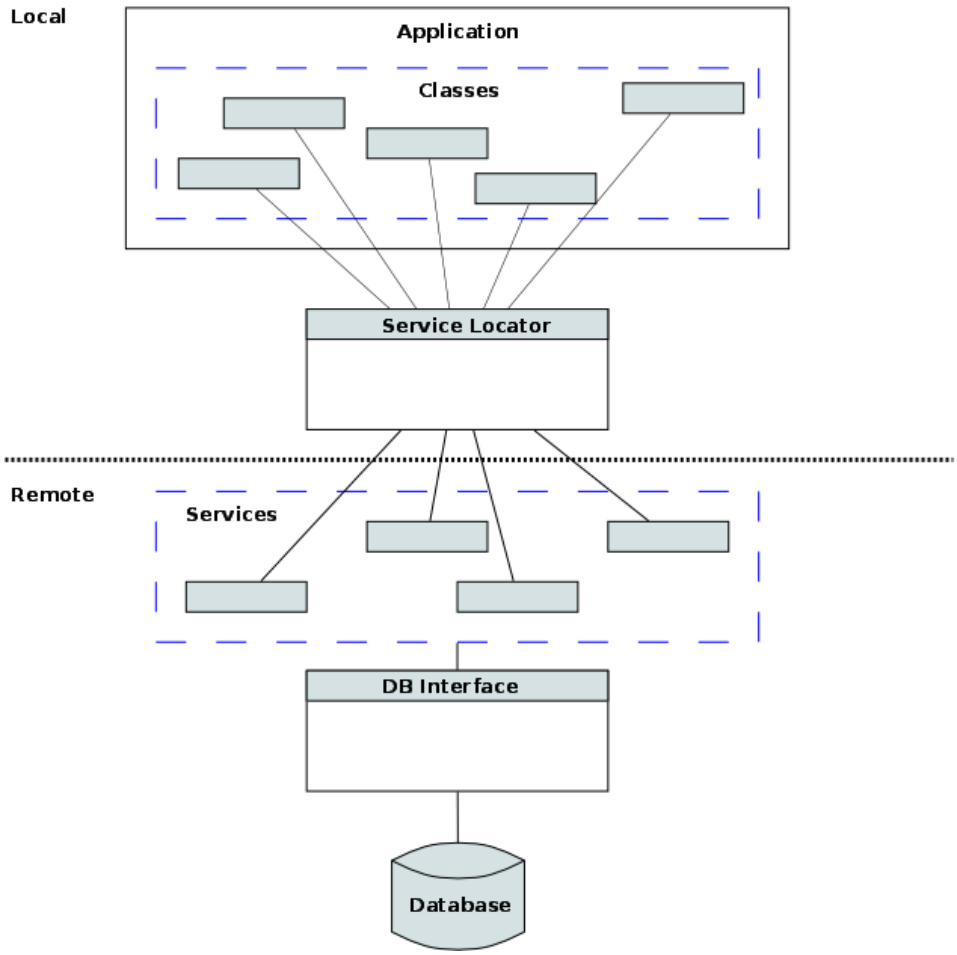

Figure 4.4: Communication architecture

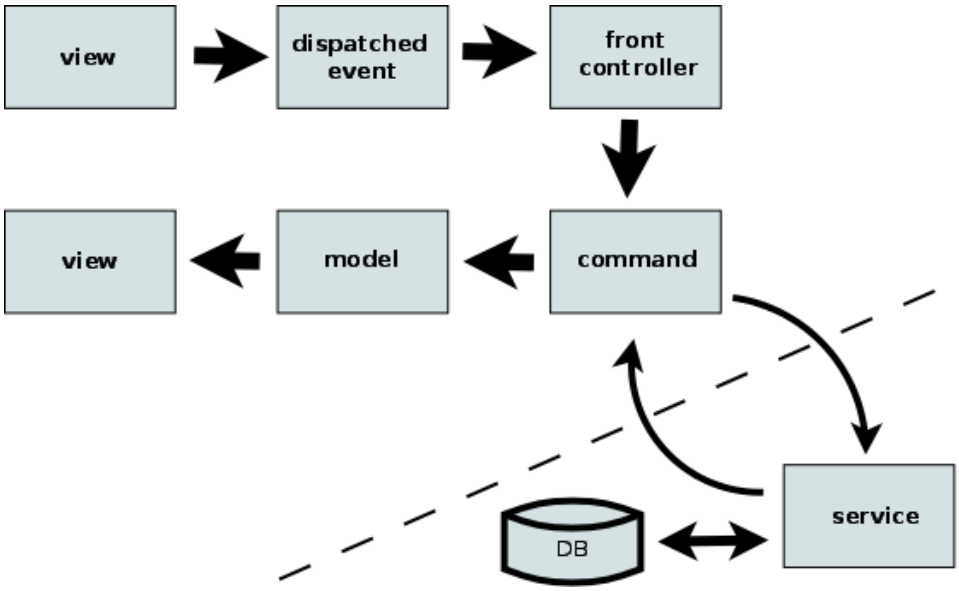

Figure 4.5: Event flow

3. front controller receives an event and dispatches a command that is mapped to that event

4. the command is executed, if required it connects with an appropriate service on the server (which probably connects to a database to exchange information). Next the command changes appropriate state in the model

5. the model updates all required values

6. the view gets updated as a result of being bound to the model

This approach however quite complicated, permits dividing an application into several independent layers, resulting in more consistent and modular code.

4.5.3 Asynchronous event processing

Numerous operations executed at once by client require asynchronous communication. This is provided by asynchronous *RPC* calls. In *Flex* application a service can be called from multiple locations, at different time. A programmer needs to take care of it. A service that is being still processed is not cancelled and is not processed from the beginning in case of another - simultaneous call. That is why some care needs to be taken during programming phase not to overlap various calls.

4.5.4 Server scalability

Although an application is currently aimed to work on only one server, some efforts have been undertaken in order to facilitate a migration in the future. The configuration file called "services-config.xml" is used to define remote access to available services, which can be located on various servers. Also some programming techniques have been used to enable a synchronisation between servers. Apart from storing path to each image in the database, an image is stored also as a "blob" type. Next when user wants to read it, an image is downloaded to appropriate server (if had not been downloaded previously). So, when user uploads a photo in reality its is inserted into a database and downloaded only to the current server. Next when someone wants to read it from different location, a file is copied to the server form a database and then displayed on a clients machine.

4.6 Compability

The application was primary intended to work on both *PC* and *PDA* hardware platforms. Although there is adequate runtime environment to run the application on both platforms, the problem raised due to selection of *Remote Procedure Calls* as a communication standard between a client and the server. The question was if I should change a method of communication with server or abandon *PDAs* compability. The selection was obvious, because good programming manner and high efficiency seemed to be more important. Furthermore application should be redesigned to meet needs of *PDAs*.

Referring to the software, there are no restrictions. *Flex* applications can be deployed on all major browsers, desktops, and operating systems. The only requirement is installed *Flash Player* greater or equal to 9.

4.7 Internet Application features

Although *Flex* is used strictly to create web application, a behaviour of programs created in this technology resemble more functionality of a desktop application. That is why many users have concerns if all required features of Internet application are met. The most common concerns are listed below.

4.7.1 Deep linking

Although avoiding constant page refresh using *Flex* application has many benefits, the main problem is that application's navigational state is not coupled to URL. This means that the address displayed in navigation bar does not change with change of application state. User cannot bookmark a page containing interesting picture and email it to a friend. Such functionality is highly desirable in case of *styleBook* application. Fortunately *Flex* is

equipped with the *BrowserManager* class that handles deep linking. This class uses methods in the HTML wrapper's JavaScript to handle events, update the browser's address bar, and call other methods. Another class, *URLUtil*, is provided to make it easier to parse the URL as you read it in your Flex application and write it back to the browser[11]. A special controller has been prepared that is responsible for preparing and receiving URLs directly pointing to user photos. Once received address it changes application state to appropriate, loads whole gallery and displays photo.

4.7.2 Localization

Nowadays developing a multilingual application is a must. That is why *styleBook* was equipped with support to handle different language versions. Currently the original language is English, but later on we are planning to add translations for Basque, Catalan, Polish and Spanish.

To make different language versions of application, it is needed to create special resources. Later they might be either compiled as bundles into the application or compiled into resource modules and loaded at run time (the main application is smaller). I have opted for the first approach, in theory worse, but perfectly suited for only 4 languages and small amounts of text.

The resources are loaded using special helper class, thus changing this functionality in the future will not be a problem.

4.8 Security

Providing appropriate level of security is extremely important during development of software. Programmers have habits of paying little or no attention to this matter what results in security flaws of application. Safety of user data, lack of unwanted spam or phishing attacks is a need, that should be provided to the end user. Outages, downtime's, traffic overload although important, do not exposure users data to be intercepted by an intruder. The architecture of *styleBook* requires providing security in both layers: called a little bit ambiguously client-side and server-side tiers.

4.8.1 Flex related security

Developing *Flex* or *Flash* application some care is also required. *Flash Player* runs inside a sandbox [5] and is composed of specific right permissions. Figure 4.6 shows a layered access which imposes that user has the overall rule for permissions and settings for the local *Flash Player*. On the other hand sandbox security system does not allow a malicious programmer

[5] A sandbox is a logical security grouping that Flash Player uses to contain resources in an environment. It will determine what sets of data as well as what operations are available to that particular Flash application[5].

to capture information that might pose a risk to security or privacy. Other features provided by *Flex* are: SSL encryption between client and server, no data is sent from users microphone or camera without a permission and there is an ability to disable storage of information for any domain.

Despite all this characteristics user data might be indirectly posed to risk, because of security holes in a Internet application they are running. It is important to remember that it is very easy to decompile *Flex* application. That is why I am not storing any sensitive data on the client side. Further more many programmers that create *Flex* - *PHP* tandem applications expect to receive and return clean data. See next section 4.8.2 for more details.

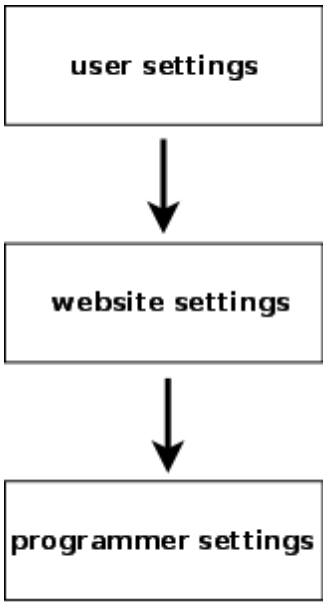

Figure 4.6: Permissions structure

4.8.2 PHP related security

There are many possible ways to attack an Internet application. Starting from Cross-Site Scripting (XSS), SQL Injection, Shell Injection, Code Injection, Session Injection, Session Poisoning, etc.

Flex has a strict security to prevent XSS. By default *Flex* functions are not callable by *HTML* scripts. Anyway all input is sanitised before saving it in database using *PHP* function “htmlspecialchars()”. Some malicious code can also be transmitted in form of an image. That is why all pictures that are uploaded by users are created from the beginning and copied pixel by pixel so that no unwanted content such as for example *JavaScript* code is transmitted.

Another, probably the most popular and dangerous attack is called SQL Injection. It is

based on inserting a modified code into database. To avoid this risk I am using prepared statements so parameters are bound after the statement is compiled (*SQL* logic is separated from the data being supplied). Furthermore a special class for communication with database has been created. One of its task is to filter data from the SQL queries (using *PHP* function called "mysql_real_escape_string()" [6]).

The same class is used to clean shell commands (using a function "escapeshellcmd()"), which improper use might lead to extremely dangerous attack called Shell Injection. The rest of attacks such as HTTP Response Splitting, Directory traversal, Session fixation or Session poisoning are not a case as I am not using programming techniques that might lead to this problems.

4.8.3 Encoding photo name

The current architecture of servers and the way in which *Apache* server works has some shortcomings. *Apache* should obtain "rwxr-xr-x" (755) permissions to the directory containing files that can be accessed by users. Having this in mind each user can view files of other users assuming that a direct link to these photos is known. Knowing one link, other can be easily guessed.

To avoid such dangerous situations I am creating a name of a photo according to the following schema:

```
MD5(time() + 'secret_string').
```

Basing only on MD5 obtained from current time, could allow a malicious user to calculate this MD5, check if is the same and compromise the security. To make it even harder, apart from adding a special string I am adding name of a photo at the beginning. The resulting name might look as follows:

```
my_super_look.JPG_2382e4bfd877964b2e4a341bffdb73c5.jpeg
```

This approach makes it impossible to find an appropriate link. Have look at section 4.9.3 to find out how photo directories are created, which provides even more security.

4.9 Directory structure

While developing medium and large size application, an act of creating appropriate directory structure is an important task. Having selected *Cairngorm* framework, this structure was already suggested to some extent for a source directory, nevertheless modifications have been done in order to improve it.

[6] Here is explained why is not a sufficient solution having a security issue: http://shiflett.org/blog/2006/jan/addslashes-versus-mysql-real-escape-string

4.9.1 General schema

The main project directory has been divided into the following parts:

- .settings - contains the preference settings for a Flex Builder project
- assets - used to story all images utilised in application together with style sheets.
- bin-debug - contains all of the files that are necessary to publish an application to the web
- config - place where services are configured, also some general application configuration goes here
- documentation - all documentation, plans, schemas, papers and learning materials go here
- externals - a place for any external data written by the server. Here are stored for example all batch files.
- html-template - additional files used by specific Flex features, such as deep linking or Flash Player detection. Flex Builder uses these files to generate an HTML wrapper for your SWF file.
- libs - directory used to store all additional libraries used in project
- locale - translations go here
- src - source code directory, described in details in section 4.9.2
- trash - this directory is used only for commodity. Instead of deleting packages, they are moved here. So if theirs lack causes some unresolved dependencies, they can be easily recovered.
- upload - directory used to store uploaded by user photos. See section 4.9.3 for more details.

See picture 4.7 for sample piece of directories.

4.9.2 Source directory

Source directory is divided into “amfphp” directory containing all server code and “com.masa.stylebook” directory containing *Flex* code. First one is later divided into “browser” that contains special tool used to debug services, “core” that contains all *AMFPHP* libraries and “services” which contains all services, value objects and helper classes. The second directory is divided into directories containing code of main, client, hairdresser and boss modules, application specific configurations and utilities (helpers).

Main, client, hairdresser and boss applications have a Cairngorm specific structure:

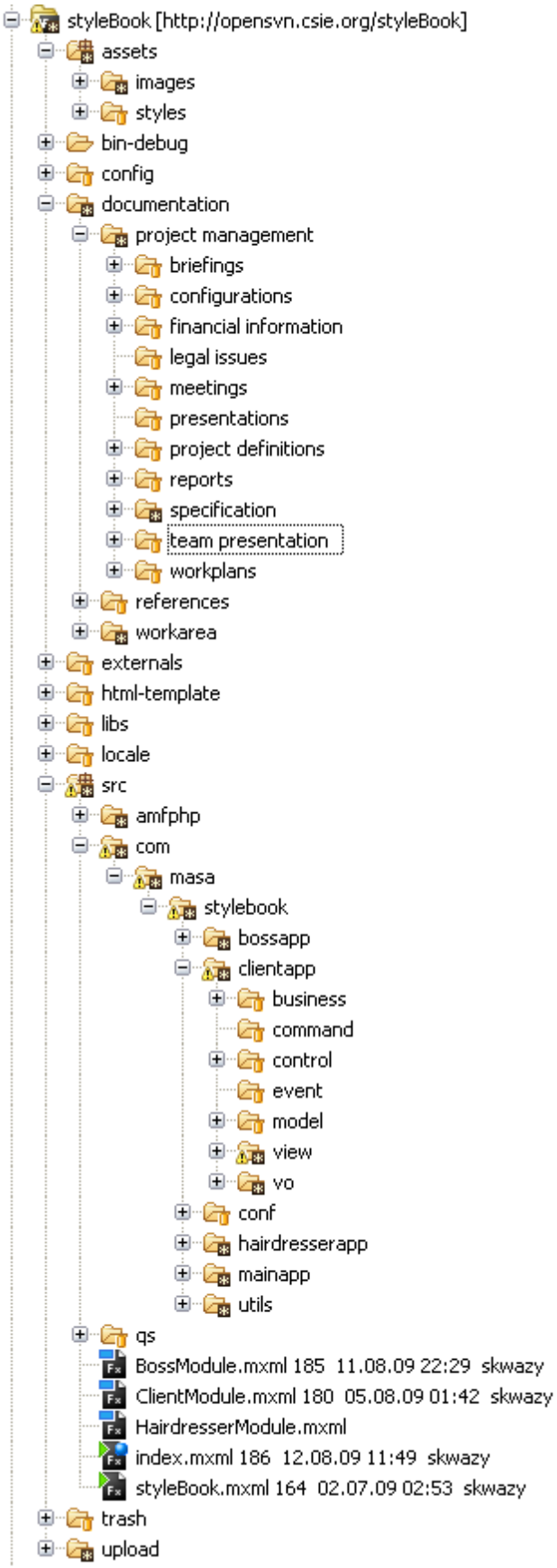

Figure 4.7: Main directory structure

- business - all business logic is put here. This directory holds also definition of services used in a given module.
- command - commands that are called by Front Controller
- control - contains main controller of a application
- event - holds all of the custom events for the application
- model - contains global Model Locator and some smaller models that hold specific data
- view - components of view are stored here
- vo - holds Value Object objects that serve as primary sources of information exchange

4.9.3 Upload directory

Upload directory is a very specific one. It is a place where are stored all user data, currently only photos of a hairstyle. Having in mind quite big and altering amount of new users, it is crucial to design this structure in an appropriate way. It is important to remember that listing a directory containing an amount of photos or subdirectories exceeding 1000, might be resource consuming. Many applications faced this problem and required redesign, because the request handling has been taking too much time.

I have decided to create directories basing on a current date. So main user directory is created according to the following formula:

```
upload/YYYY_WW/user_id
```

which for example results in the following input:

```
upload/2009_27/5996
```

meaning that user created an account in 27TH week of year 2009 and has id 5996. This approach ensures that there will not be too many user directories in the main directory. If this count rises to more than 1000 in given period of time, the naming convention of directory could be modified, for example switching from weeks to days of week (in simplicity from 52 to 365 subdirectories).
The pictures of a user are saved in a similar way:

```
PATH_TO_USER_DIR/YYYY_MM/photo.extension
```

which for example results in a following input:

```
PATH_TO_USER_DIR/2009_08/IMG_3371.JPG_6caf0d4b5dd8d4d56ee1eaec36add6eb.jpeg
```

So the absolute path to the photo would be the following:

```
/upload/2009_27/5996/2009_08/IMG_3371.JPG_6caf0d4b5dd8d4d56ee1eaec36add6eb.jpeg
```

This approach as previously mentioned is efficient and in case of excessive amount of either users or photos, more suitable naming convention could be applied. To enable such change, whole path (after "upload" word) is saved in database.

To see more details about photo name creation policy, see section 4.8.3

Chapter 5

Application

In order to separate specific functionality of the software into logical pieces, the code was divided into *Main Application*, *Client Module*, *Hairdresser Module* and *Boss Module*. This aims to improve management of code, reduce size of the application that is being loaded and make it load faster.

5.1 General overview

As a consequence of architecture 3.5 selected for *styleBook*, the *Main Application* is the key element, indispensable for proper functioning of an entire system. This piece of code is responsible for exchange of information with modules. Although modules contain their own functionality, they need from time to time to exchange data with this element.

5.1.1 Design

Providing nice and up to time design and usability was one of the most important questions during development of *styleBook*. That is why many meeting have been arranged in order to determine all important details. The main aim was to build Reach Internet Application, thus the choice of technology used to develop client-side application was also influenced. As mentioned in the team task delegation part 1.4, design matters were not done by the author of this thesis, however I had influence on planning of look & feel.

The following picture presents *styleBook* logo: Name *styleBook* originated from two

Figure 5.1: Stylebook logo

words: “style” - related with a topic of this application and “book” - connected to an

application 5.3.3 that is used to view photos, which simulates in great details behaviour of a normal book. Furthermore combination of these two words is easy to remember and can be simply associated with our service. Although a similarity of names with *Facebook* might cause some scarce misunderstandings.

5.1.2 Conflict resolving algorithm

Conflict is a situation when two or more clients have selected the same time of reserve. Although the first user is not aware of this problem, next clients that book this hour, see all details. That is why each user can select three times in order of preference to eliminate situations when the request is rejected, because other person had already reserved this moment or made a mistake and a hairdresser has to increase the time of visit and reject requests that still remain unconfirmed. An indirect reason for such situation might also be a long time of hairdressers response, who's task is to accept pending visits as fast as possible. The algorithm is the following:

1. accept all non-conflicting reservations (automatic, theoretically no hairdresser interaction required)
2. the processing of conflicts starts from the user that is first in the queue. Check first preference[1], if a time is already used or unavailable go to the second preference, repeat and if required go to third preference. If none of preferences is acceptable, user receives an appropriate notification.
3. jump to next user and repeat all steps finishing on the last user
4. if any users still stays without selection, send a message with notification

The whole process described above can be interrupted by a hairdresser (after algorithm has finished). When worker of a hair salon notices that a time selected by a user is not long enough, that additional time might be reserved. Selection might be done only of conflicting and free fields. This however might lead to another conflict, so that algorithm is run once more. After all changes have been done, hairdresser confirms them and conflicting fields change to reserved and become unavailable. The description of available states can be found in section 5.4.1.

5.2 Main Application

As the name suggests it is a main application that embeds other modules. It contains code that is being shared among all users, such as message system, notification system or login and upload component.

[1] Preference is name for time in which user would like to go to a hair salon. It is called preference till the moment of confirmation when it becomes reservation

This section is divided into several components, which are described below. They are all part of *Main Application* and are available to all kind of users. Having once described given component, I will not repeat it for each module.

5.2.1 Unregistered user view

This view is run at the beginning when user accesses our website. It provides some basic functionality described below, and after doing login it simply loads an appropriate module according to a user type.

The particular components that build this view can be noticed at picture: 5.2. Generally it can be divided into the following components:

- *Main Logo* - is a link to a main page, its location differs according to a user type
- *Main Menu* - an interactive and animated flash component that changes the state according to user action
- *Login component* - used to log user in or display welcome message depending on a state. If user has no account, registration/user data modification view is called which is visible on figure 5.3. This component is equiped in all required validators and contains an interactive city selection component.
- *Static bar* - a graphical element to improve look & feel of the application
- *Recent Hairstyles* - a component that displays recently added pictures. If more pictures were uploaded by one user, than the first one is selected. If user creates new hairstyle [2] than the default photo is displayed.
- *styleBook news* - displays news created by *styleBook* editors
- *Salon news* - a component that displays news recently created by a salon. It shows only a header, another component is opened after clicking link.
- *Widget Bar* - an interactive widget bar that contains shortcut to clock, calendar, weather and hair visits components.

5.2.2 Messages component

This component is used to send and receive messages, technical details were described in the section 4.4.3. I have opted for a grouping technique, assuming that the majority of users tend to reply to the first message and prefer to see them all joined together. This technique glues together messages that have the same parent (solution known from *Gmail*[3]

[2] In *styleBook* project hairstyle is a name for gallery that groups some pictures
[3] https://mail.google.com/

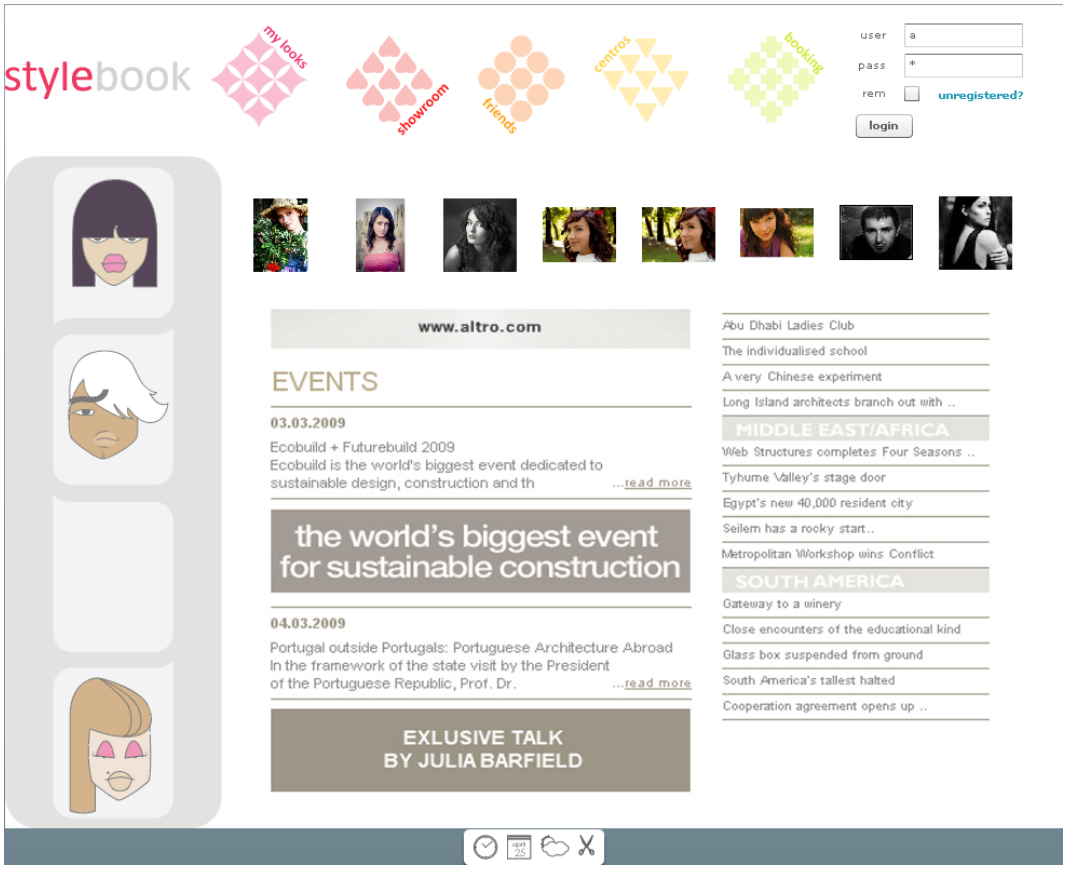

Figure 5.2: Main application view - unregistered user

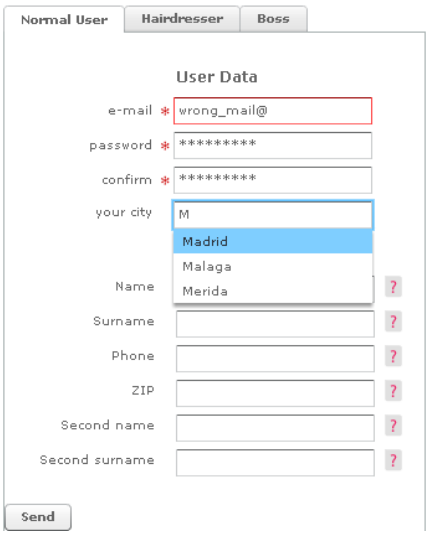

Figure 5.3: Registration box

later introduced also by *Facebook*[4] and many other services). User can only remove to trash a whole conversation. There are no restrictions posed on mail communication, every user registered in the system can send and receive messages.

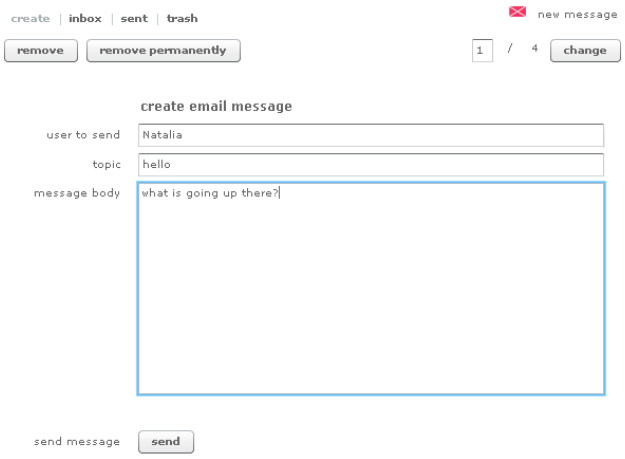

Figure 5.4: Email view

5.2.3 Upload component

Upload component presented on figure 5.5 is a flexible application that is used to upload photos to the server. User is able to select numerous photos at once, choose appropriate gallery (or create a new one) and without any worries about the size[5] upload pictures to the server. Size reduction is done on the client, so server is not loaded with unnecessary data that anyway would be lost after resize done on the server.

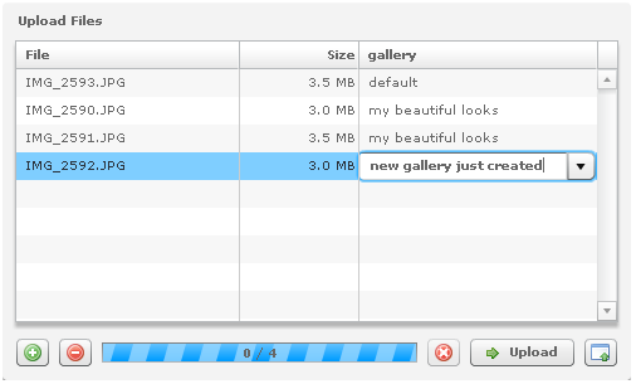

Figure 5.5: Photo upload component

[4] http://facebook.com/

[5] The *ByteArray* class that is storing information about a photo has limit of 256MB, which is equivalent to picture of 8192x8192 pixels

5.2.4 Image map component

styleBook project is primary aimed to launch in two countries: Poland and Spain. Never the less some classification of geographical location has to be applied, because the quantity of cities is too big to select them from a combobox or a list. Creating an image map for level of country and region (an administrative division) seams to be a good idea. This component is used in search of salons or selecting user location.

Figure 5.6: Image Map component

5.2.5 Notification box

Displayed in top part of user view. It scrolls down, waits some time letting user to read the message and hides automatically. It also can be opened by clicking an arrow. Used to display errors, warning and notices coming from the server.

5.3 Client Module

Client module is a part that is strictly related with social network. Although some basic functionality is already included in *Main Application*, this part contains many specific components that are not used in other modules. Figure 5.7 presents a basic view.

5.3.1 Minis

This component is integrated with client view having a position fixed to the left side. *Minis* are small components providing access to the most popular functionality that is never hidden by other window. They are divided into four components:

- *MiniUser* - displays user information and photo. It also contains current client state and a description.

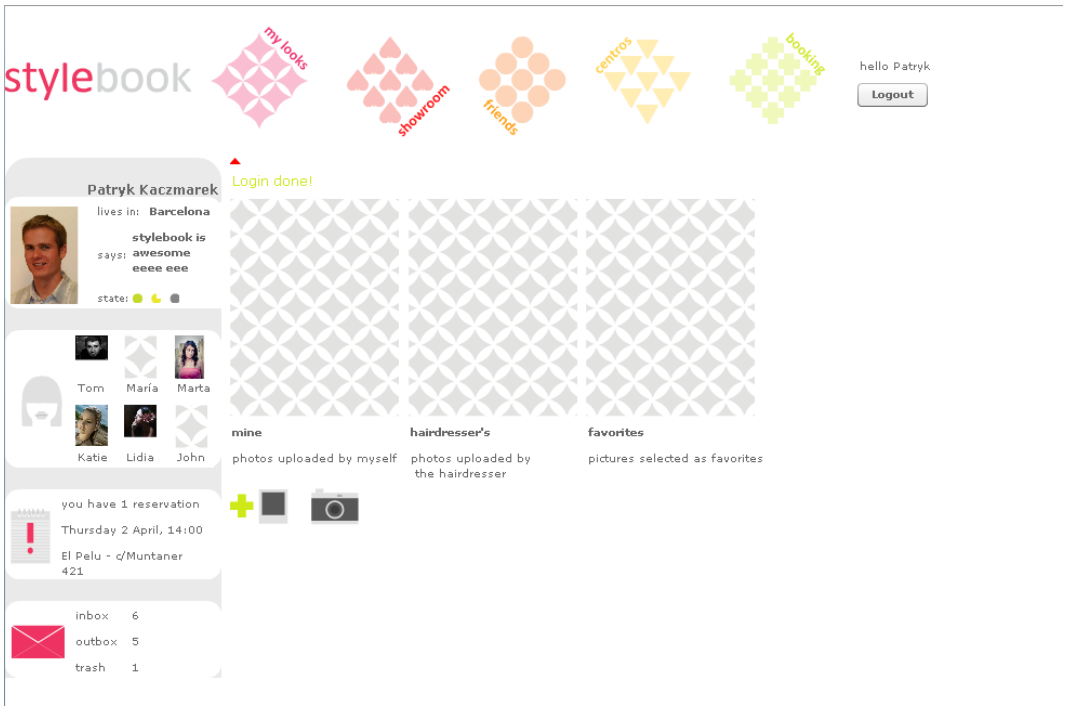

Figure 5.7: Main Client view

- *MiniFriends* - direct access to six friends chosen randomly
- *MiniReservations* - shows information about current booked visits
- *MiniMessages* - serve as a shortcut to email boxes

5.3.2 My Looks

A place to browse all available looks and hairstyles. Each hairstyle is characterised by specific permissions so the access might be restricted for some users.

There are three levels of access to a given gallery:

- public - visible by everyone
- friends - visible only by your friends
- private - nobody except the owner has access to this gallery

Generally *My Looks* component can be divided into three parts:

- *my looks* - a section that is used to store personal photos of a user. Each photo can be contained in a separate gallery (created by a user) or in the default one. User can choose any photo for an album cover. By default access to each gallery is set to "friends".

- *hairdresser looks* - no division into galleries is available at this stage. This container is used to store photos of your hairstyle that were uploaded by a hairdresser. The state of a photo can be either:
  - accepted - user has accepted this photo and moved it to a gallery. Default access is set to "friends".
  - awaiting acceptance - user has not already accepted a photo. Default access is set to "private".
- *favourite looks* - photos of others that user has tagged as favourites. The access is fixed and set to "private". Once owner of a photo changes an access to such that does not allow other user to see it, then a picture disappears from this folder.

5.3.3 Photo browser

Photo browser is the main heart of this application. Its functionality is dependent on current permissions of a gallery (owner, friend, stranger). Picture 5.8 present a *Photo Browser* available for owner of the photo. It is composed of several components:

- *FlexBook* - a component that simulates behaviour of a normal book by a page flip effect [4]. It is used to view photos in a big size. It is integrated with the *ThumbnailViewer* to switch photos displayed in that component when user changes a page.
- *ThumbnailViewer* - a component which displays thumbnails below *FlexBook*. It is used for faster navigation in a gallery.
- *Ratings* - used to rate a photo in scale 1–10. Each user can vote only once for given hairstyle picture.
- *Quick access* - is composed of four buttons. They allow to set a picture as a default, delete, add as favourite or add to the reserve.
- *Description* - used to add a photo description
- *Tags* - sets characteristics of a photo. It is later used by application that searches photos by tags 5.3.4.

This application is equipped in a special caching mechanism. At the beginning a photo thumbnail is extended to size of a big photo, while picture is being loaded. Finally when a picture gets downloaded, it is saved in cache, and a succeeding picture is being downloaded to the memory, so that user does not experience download delays.

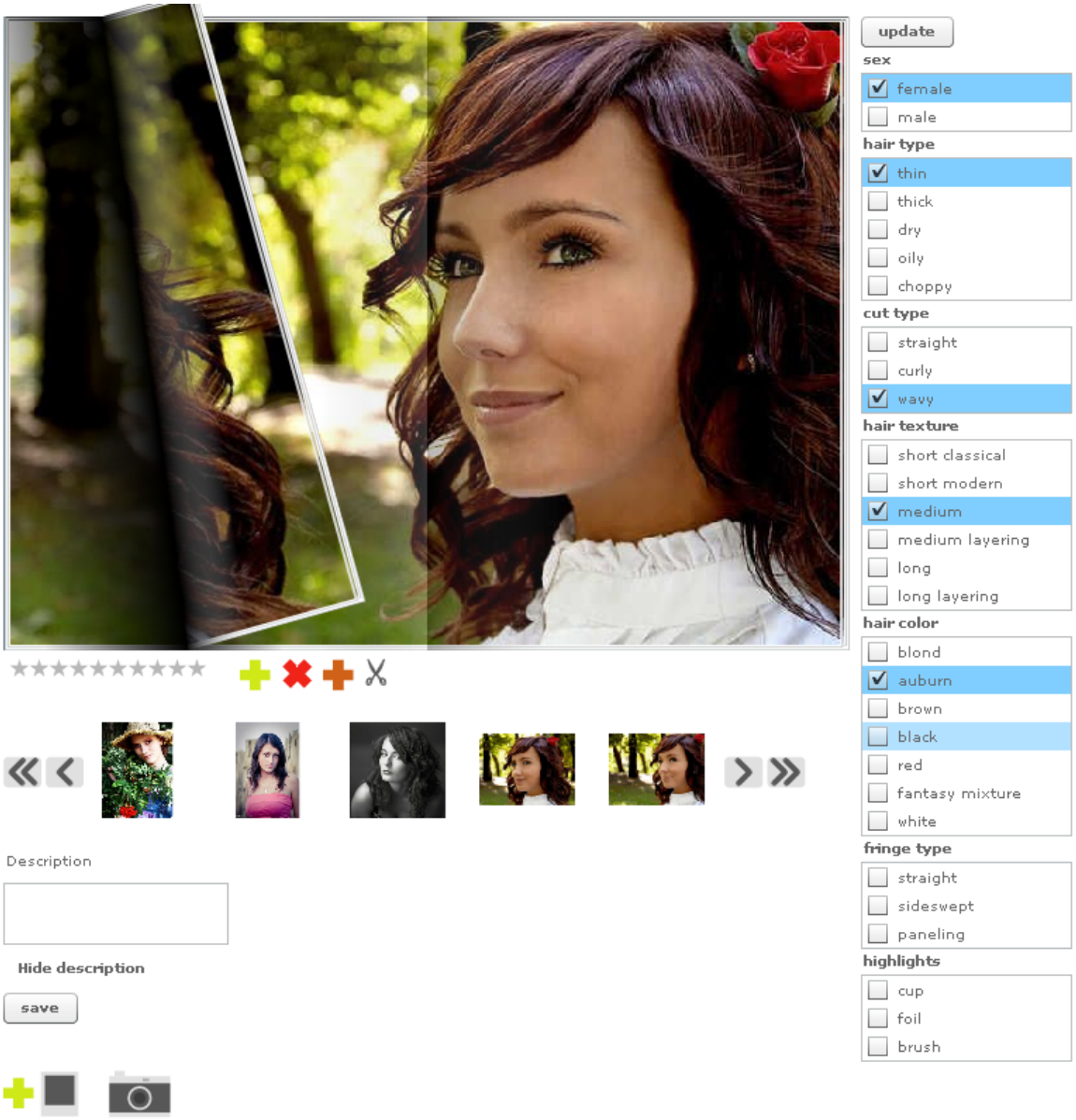

Figure 5.8: Photo browser

5.3.4 Showroom

This is a component that is used to search for interesting pictures. It is based on the idea of tags. User select one tag from each category (more details on categories can be found in model section 4.4.7) and starts search. The more tags selected, the more concrete will be the results. To make more general search some tags should be removed.

5.3.5 Friends

Friends component contains all functionality connected with management of friends. User can view all friends, search new connections by email and see all requests that are awaiting confirmation.

5.3.6 Centres

- search for centre - used to look for a hair salon. To avoid this activity, a centre can be added to favourites.
- view my favourite centres - component used to browse centres that have been visited or added to favourites by a user.

5.3.7 Booking

This part is used either to make a booking or to view current bookings. Picture 5.9 presents this component. User gets this calendar after selecting centre of his choice. There are three colours used to visualise the state of a given time period. It can be either:

- green - nobody has subscribed for this hour. The probability of reserving a visit is very high.
- yellow - somebody has already chosen this hour. The choice will depend on the algorithm 5.1.2, but might also be modified by a hairdresser.
- red - there is no possibility to book this time. Either somebody has already reserved it (and hairdresser did a confirmation) or hairdresser is unavailable in this moment.

See subsection 3.1.2 for more details on reserve mechanism.

5.4 Hairdresser Module

Hairdresser module is very restricted in comparison to the client equivalent. Its design and functionality are minimalistic not to distract the attention that should be payed towards clients not to this application. That is why apart from the common for all users components it has only one component that is responsible for management of reserves.

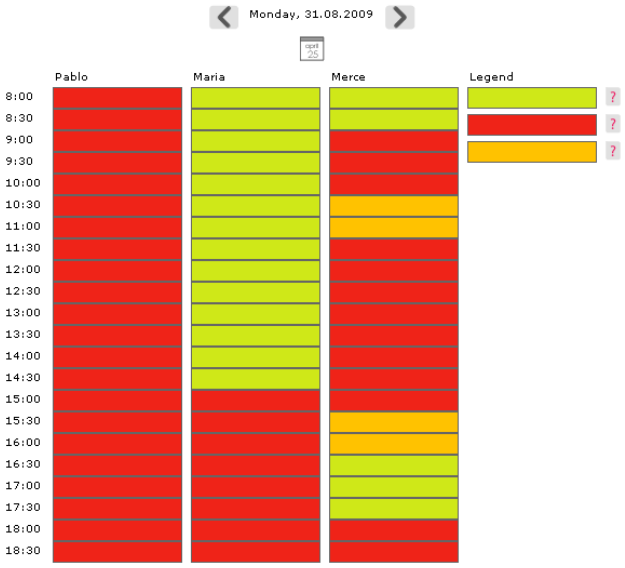

Figure 5.9: User calendar

5.4.1 Reserve Calendar

Although *Reserve Calendar* looks very simple, its is quite complicated from the programming point of view. As shown on figure 5.10 it is composed of three parts,

- a dragable table cell - users can be moved by a *Drag and Drop* technique [6] and dropped in white (free periods of time) fields.
- three action boxes - are used to create state of table cells. When user makes a pre-reserve the appropriate field turns red. A hairdresser can then click green box, which is equal to acceptance of this visit. Once accepted, a hairdresser can reject a visit, by clicking a white box. White boxes can be converted into black and the other way round.
- legend with action buttons - used to explain what does each colour mean and buttons are used either to save last modifications or to reject them

Explanation of colours used:

- white - a field is free, user can be moved here
- green - a field has already been reserved, it might be either a client from the system or someone reserving visit through a phone or in person. This field can be changed to

[6] Drag-and-drop is the ability to move graphical user interface (GUI) objects, i.e., icons and windows (and menus on some programs), by means of manipulating a mouse or other tracking device (such as a trackball, touchpad or pointing stick). [17]

white when hairdresser moves a client to different hour or to black meaning that the worker will not be available in this moment (a client gets then appropriate notification in both cases).

- red - to authorise, which means that someone wants to reserve this time,
- orange - this field is conflicting, many users have marked it.
- black - out of work - means that a hairdresser is not available in this moment

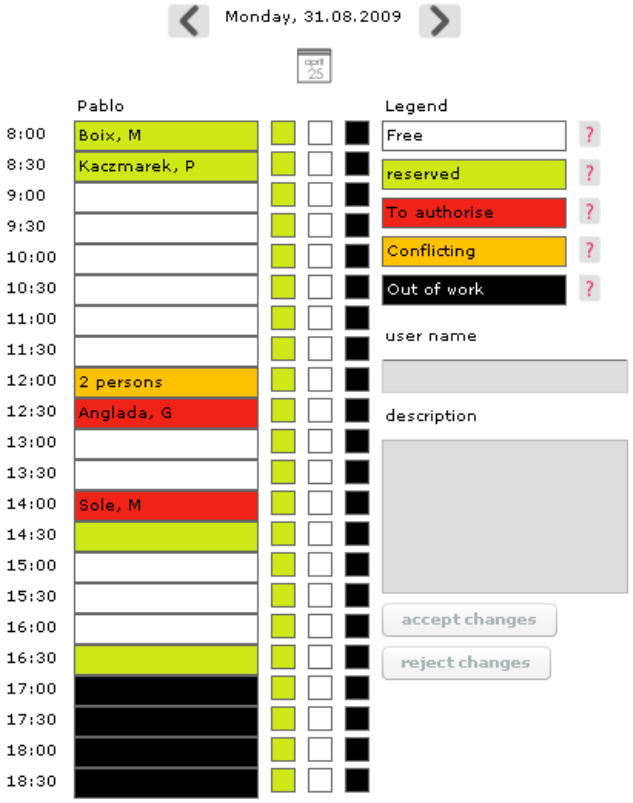

Figure 5.10: Hairdresser calendar

5.4.2 Conflict resolver

This component will be created in the future to resolve conflicts between clients that have chosen the same time. Currently this is done automatically what might not be the best solution.

5.5 Boss Module

This module is also uncomplicated and contains only functionality related to management of a company and corresponding hairdresser salons. It can be divided into the following views:

- *Company management* - this component permits modification of basic company [7] data. The boss can change name, description, main photo, manage departments and hairdressers. See picture 5.11.
- *News* - a place to manage all company news. Currently there is only one template available, which is composed of news topic, content and one photo. See picture 5.12

Figure 5.11: Main Boss view

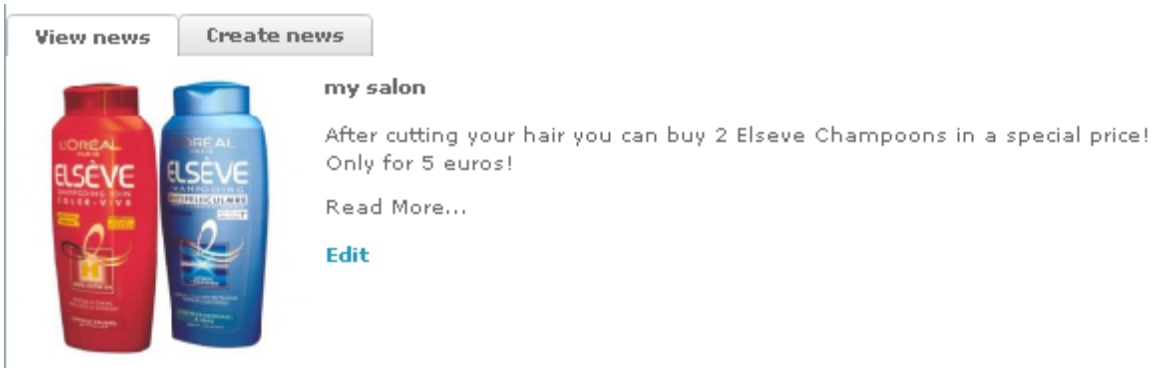

Figure 5.12: News view

[7] Company is a name used in this thesis for the main department or the head office

Chapter 6

Testing

The quality of a developed system requires constant inspections. A programmer cannot conclude that a software is free of errors. That is why systematic code inspections have been done, which resulted in smaller amount of system failures.

6.1 Code inspection

Code inspection is a method of a systematic source code analysis that has one main objective: find as much errors as possible. This search has to be done by a third party, a person that did not implement a specific part of a system. The rotation of people is important, because it is easy to get trapped and not see one's own errors. The way of thinking gets directed to a specific, dependent on a person, way of realisation of a specific functionality.

Unfortunately in case of *styleBook* development, the code could only be inspected by one and always the same person. Inspections were normally carried out before implementation of given function. Although this approach is not perfect, it was a productive method of error searching. Furthermore currently developed application is only a prototype, which would be analysed thoroughly in the future by bigger group of programmers and testers.

6.2 Look and Feel testing and debugging

Although seams to be an easy task, much time was consumed on laying out views. Very useful turned out to be *FlexSpy*[1] a component that lets programmer browse all visual components and dynamically change properties and styles for those components in real time. This tool makes debugging easier and

[1] http://code.google.com/p/fxspy/

6.3 Unit tests

This is a validation of functionality of single functions. This testing should be done at the same time as implementation of a system. An error that is found rapidly is easier understand and to eliminate. I have done it using a method of "white box" meaning that I had access to implementation of a tested function. Carrying out tests was possible thanks to a special testing framework provided by *Flex* called *Flex Unit* and carried out mainly in commands because they are core in *Cainrgorm* development. Testing helped me to refactor code and reduce complexity.

6.4 System tests

The aim is to compare a real functionality with requirements. I was testing whole modules checking if an application is catching and handling error in an expected way after introducing incorrect input data.

6.5 Acceptance tests

This tests were carried out in presence of all team members. Their aim was to show given functionality, analyse it and finally accept.

Chapter 7

Conclusions

The *styleBook* system is an innovative solution designed strictly to serve certain needs. The main purpose of this project is to gain benefits such as time, money, human resources and simply to give fun to end users. The primary objective of this project is met - the project significantly eases a great majority of procedures incorporated in a hair salon reserve activity. Apart from it there are several other profits that *styleBook* delivers to a user of thus system, e.g. social network functionality that extends significantly the exchange of information in style matters field.

7.1 Completed project objectives

Not only does the system fulfill all of the requirements set by the clients of this enterprise in the preliminary stage, but it also successfully complies with the global project objectives set up during the analysis stage. The following list contains a summary of the completed requirements that are included in the specification:

- development of a system specialized in management of hairdresser visits
- creation of a specific algorithm, used to automatically resolve conflicts resulting from the same time of reserves
- elaborating an efficient, highly interactive social network part that is composed of many interesting components
- a template to create a dynamic content that is used as a presentation layer of hair salons

7.2 Possible improvements

Although development of project was composed of systematic increments, some delays were encountered. At the first stages the graphical design was a leading part and was somehow slowed down by lack of all requirements and small idea about the technology that will be applied. On the other hand having once entered into the coding phase, roles changed. Often altering requirements imposed a change of design and generally requirements. Finally I had to wait long periods to get required images and skins. This lead to incomplete development of some less important features. To improve this matter, more time should be spent in a requirement phase.

If I were to repeat a design phase and selection of required technologies, nowadays I would opt rather for *Zend* framework instead of *AMFPFP framework*, which is currently not being developed. I have followed opinion of specialists and however the choice is good, it could be better. As as result the learning curve would be more abrupt, because complexity of *Zend* framework is higher.

7.3 Future concepts

Some additional functionality will be probably implemented in the future - after complete and detailed analysis of the current system. Some components that would be very useful, for instance *Conflict resolver* - a tool used to resolve conflicts in a more logical and including more interaction with a hairdresser way. This application would probably take more arguments apart from priorities and time of reserve into the consideration such as opinion about given client. Further more it would be very useful to integrate *styleBook* with a *OpenInviter* (more details explained in appendix A) a service used to import contacts providing an email address.

7.4 Gained experience

Writing this thesis has resulted in many new skills related to computer science. I had finally a stimulus to learn *Flash* technology, which always seamed to be strange to me. Further more this project resulted in gaining a big knowledge in fields of *Flex* related technologies like *Cairngorm* - a framework that is most commonly used by programmers I have also discovered many tools written by *Adobe* used to create graphical design. Apart from profits seen from a strictly technical point of view, this thesis has developed a better understanding of social networks. Much analysis has been carried out in this field and I have learnt many tricks that are used to construct such networks. Also some knowledge related to style matters was acquired.

Appendix A

Term explanation

Tips and tricks to help hairdresser choose a good hairstyle

There are several points that should be followed if you want to change a hairstyle. They will facilitate job of a hairdresser.

- take a picture - can be yours or picture of celebrity
- go with clean and natural state hair - make your hair seem most similar to your natural one. If you have got curly hair, do not make it straight, etc.
- dress right - dress up in your normal manner. Do not wear clothes that you put on only for special occasions
- add appropriate make up - put a make up that a hairstyle should suit to

OpenInviter

Open source OpenInviterTM (Open InviterTM) is an free import contacts (addressbook) script from email providers like Zapakmail, Mail.com, India, Pochta, Rambler, Apropo, Canoe, Inbox.com, Walla, Nz11, Interia, GMX.net, Evite, Meta, Freemail, IndiaTimes, Aussiemail, Lycos, Grafitti, Virgilio, 5Fm, Clevergo, KataMail, AOL, Abv, Mynet.com, O2, Techemail, Live/Hotmail, FastMail, Gawab, Terra, Bigstring, Popstarmail, Atlas, Mail2World, Libero, Inet, Care2, Mail.in, YouTube, Wp.pt, Mail.ru, Web.de, Netaddress, Yandex, OperaMail, Kids, Uk2, LinkedIn, Azet, Bordermail, GMail, Yahoo!, Doramail, Rediff, Sapo.pt, Hushmail or social portals like Flickr, Friendster, Mycatspace, Bebo, Cyworld, Hyves, Eons, Livejournal, Vimeo, Konnects, Lovento, Brazencareerist, Hi5, Bookcrossing, Multiply, Flingr, Kincafe, Mydogspace, Koolro, Twitter, Fdcareer, Plazes, Badoo, Xuqa, Meinvz, Xanga, Famiva, Plurk, Vkontakte, Plaxo, Ning, NetLog, Mevio, Friendfeed, Faces, Tagged, Perfspot, Xing, Skyrock, Facebook, Orkut, Motortopia, Flixster, MySpace, Last.fm. This

contacts importer script is integrating with content management systems (aka CMS) like Social Engine, Joomla, joovili, Atmail5, SimpleMachines Forum (SMF), phpFoX, PhpBB, PHPMELODY, Joomla1.0, Buddy Zone, Drupal, Boonex Dolphin, Wordpress, PunBB, JamRoom, nowFire, myBB, phpizabi, RoundCube, Dating Pro, symfony, jamit job, vBulletin, Vwebmail. Open Inviter is written in PHP 5 (no database required but cURL or wget required) and running on any webserver (tested on Apache) offering advanced tell a friend features. OpenInviterTM is a free self hosted solution that does not use a third party gateway (or API) to import contacts.[16]

Bibliography

[1] Software Engineering Standards Committee of the IEEE Computer Society. Ieee recommended practice for architectural description of software-intensive system, 2000. [on-line] http://www.enterprise-architecture.info/Images/Documents/IEEE%201471-2000.pdf.

[2] Wikipedia. [on-line] http://en.wikipedia.org.

[3] Adobe. Cairngorm developer documentation. [on-line] http://opensource.adobe.com/wiki/display/cairngorm/Developer+Documentation.

[4] Joe Berkovitz, David Hassoun, Andrew Trice, Todd Prekaski, Jun Heider, Joseph Balderson, Tom Sugden, and Peter Ent. *Professional Adobe Flex 3*. Wiley, John & Sons, Incorporated, 2009.

[5] Charles Bihis. An objective look at the flash player 9 security model. *Technical Evangelist Intern*, 2006.

[6] Karol Bonenberg, Piotr Gadzinski, Patryk Kaczmarek, and PawełWarczynski. The integrated management system for mass sports events. Master's thesis, Poznan University of Technology, 2008.

[7] Adobe Cairngorm. Cairngorm docs. [on-line] http://www.cairngormdocs.org/tools/CairngormDiagramExplorer.swf.

[8] CollabNet. Version control with subversion. [on-line] http://svnbook.red-bean.com.

[9] John Crupi, Dan Malks, and Deepak Alur. *J2EE. Wzorce projektowe*, volume 2. Helion SA, 2004.

[10] Nicole B. Ellison and Danah M. Boyd. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, (11), 2007.

[11] Flex. Livedocs. [on-line] http://livedocs.adobe.com/flex/3/html/help.html?content=08_Dates_and_times_3.html.

[12] Bruno Goncalves and Jose J. Ramasco. Human dynamics revealed through web analytics. *Physical Review E 78, 026123 (2008)*, 2008.

[13] Paul Hudson. *PHP in a Nutshell*. O'Reilly Media, Incorporated, 2006.

[14] Bob Hughes and Mike Cotterell. *Software Project Management*. McGraw-Hill Publishing Co, 2004.

[15] MySQL. Mysql 5.1 reference manual. [on-line] http://dev.mysql.com/doc/refman/5.1/en/index.html.

[16] OpenInviter.com. What is openinviter. [on-line] http://openinviter.com/index.php.

[17] The Linux Information Project. Drag and drop. [on-line] http://www.linfo.org/drag-and-drop.html.

[18] Chris Schiflett. *Essential PHP Security*. O'Reilly Media, Incorporated, 2006.

[19] Arie van Deursen and Ali Mesbah. An architectural style for ajax. *Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture (WICSA'07). IEEE Computer Society, 2007*, 2006.

[20] Ted Zimmerman. amfphp. [on-line] http://www.amfphp.org/.